Apple Pascal Trickkiste

Ron DeGroat u.a.

PUFFIN CHAR-EDIT RECOVER DECODE TEXT-INFO PBMENU LCA PATCHOS HUFFIN CODE-MAP ZAP PMU

pandabooks

Apple Pascal Trickkiste

Apple Pascal Trickkiste

Ron DeGroat u.a.

Pandabooks, Berlin

Titel der englischsprachigen Originalausgabe:
Call-A.P.P.L.E. In Depth - All About Pascal

Deutsche Übersetzung: Bodo Meseke

ISBN 3-89058-030-0

Printed in West Germany

Inhalt



Zu diesem Buch ist eine beidseitig bespielte Diskette mit allen im Buch aufgeführten Programmen erhältlich.
Sie können die Diskette über Ihre Buchhandlung beziehen oder gegen Voreinsendung von DM 50,- (V-Scheck, bar) bei:

Ron DeGroat

Das Pascal- + BASIC-System

Einsatzmöglichkeiten

Bisher waren Pascal und BASIC nicht miteinander kompatibel. Doch über die in diesem Artikel beschriebenen Routinen kann Pascal mit dem im ROM festgelegten BASIC Ihres APPLE-Computers operieren und interagieren. Hier eine Liste der Möglichkeiten, die Ihnen das Pascal + BASIC-System bietet:

Ausführen von Pascal-, BASIC- oder DOS-Operationen, ohne neu booten zu müssen.

Speichern von BASIC-Programmen auf Pascal-Disketten.

Ausführen von BASIC-Programmen unter dem Pascal-Betriebssystem.

Aufruf von BASIC-Programmen unter Kontrolle von Pascal-Programmen.

Speichern von mit BASIC entwickelten Assemblerprogrammen auf Pascal-Disketten.

Aufruf von BASIC- und Monitor-ROM-Routinen unter Pascal.

Benutzung des BASIC-Monitors unter Pascal.

Verwendung des UCSD-Assemblers zur Erzeugung von Maschinencode, der auf DOS 3.3-Disketten gespeichert und vom BASIC benutzt werden kann.

Einfache Programmierung eigener Pascal + BASIC-Mischprogramme.

Systemanforderungen:

APPLE-Pascal (II.1 oder 1.1)

DOS 3.3

Wenn Sie über das Pascal + BASIC-System verfügen, wird es unnötig, Programme, die schon für BASIC entwickelt worden sind, ein zweites Mal zu schreiben. Die meisten von ihnen lassen sich auf Pascal-Disketten speichern und mit dem Pascal-Betriebssystem benutzen. So habe ich zum Beispiel unter BASIC dreidimensionale Hi-Res-Grafikroutinen entwickelt, sie auf Pascal-Disketten gespeichert und dann in Pascal-Programmen verwendet.

Das Herz des Pascal + BASIC-Systems besteht aus der Unit BASICSTUFF und dem Hybridprogramm PBMENU, das auf diese Unit zugreift.

Verzeichnis der Listings

Listing Nr.1
PBMENU: Ein Pascal-Hybridprogramm, mit dem fast jede Routine aus BASICSTUFF interaktiv benutzt werden kann.

Listing Nr.2
BASICSTUFF: Eine Unit, die es nach ihrem Einbau in die SYSTEM.LIBRARY gestattet, mit dem ROM-residenten BASIC zu interagieren, das sich gleichzeitig im Speicher befindet.

Listing Nr.3
PBPROC: Externe Prozeduren, die von BASICSTUFF benutzt werden.

Listing Nr.4
DOS32K: Ein Pascal-Programm, das DOS 3.3-Masterdisketten in 32K-Disketten konvertiert, damit PBMENU beim Laden von DOS nicht überschrieben wird.

Wie man PBMENU benutzt

PBMENU ist leicht zu bedienen, da seine Befehlsoptionen entweder selbsterklärend sind oder BASIC-Anweisungen entsprechen.

Bildschirmausgabe von PBMENU:

PASCAL + BASIC: BEFEHLSMENÜ

BRUN
BSAVE
BLOAD
RUN
SAVE
LOAD DOS
CALL
ROM CALL
QUIT

RECHTS- UND LINKSPFEIL ZUR CURSORBEWEGUNG

LEER- ODER RETURN-TASTE FÜHRT GEWÄHLTEN BEFEHL AUS

LOAD DOS - Lädt das DOS von einer Pascal-Datei in den Hauptspeicher, bindet es in das Pascal-Betriebssystem ein und ruft das jeweilige, ROM-residente BASIC auf. Diese Option geht davon aus, daß das DOS bereits auf einer Pascal-Diskette unter dem Namen "DOS.3.3.BIN" gespeichert wurde (später mehr darüber). Wenn man diese Option wählt, wird das DOS in die Adressen $5600 bis 7FFF geladen und HIMEM: automatisch gesetzt, um es vor einem Überschreiben zu schützen.

ROM CALL - Funktioniert wie der normale CALL-Befehl mit dem Unterschied, daß das ROM vor dem Sprung eingeschaltet wird. ROM CALL rettet außerdem bestimmte Pascal-Informationen, die überschrieben werden könnten, tauscht die Pascal-Zero-Page gegen die BASIC-Zero-Page aus und stellt beim Rücksprung den alten Zustand wieder her.

SAVE und BSAVE hängen automatisch das Suffix ".BAS" oder ".BIN" an den vom Benutzer angegebenen Dateinamen. RUN und BRUN stellen das entsprechende Suffix automatisch zur Verfügung. Alle anderen Befehlsoptionen funktionieren genauso wie ihre BASIC-Entsprechungen.

Mit SAVE oder BSAVE erzeugte Dateien haben in ihrem ersten Block einen Record, der PBCODEINFO genannt wird und wichtige Informationen enthält (vgl. Listing Nr.2). Diese Informationen werden von RUN und BRUN benutzt und enthalten Angaben wie die Startadresse der Routine, die Codelänge und eine komplette Kopie der Zero-Page zur Zeit der Speicherung.

Das PBMENU-Programm teilt den Hauptspeicher so auf, daß der BASIC-Teil des Systems einem 32k-Rechner entspricht. Wegen dieser Speicherbeschränkung kann es vorkommen, daß manche Programme für das Pascal + BASIC-System zu lang sind. Mit Hilfe von ein paar Tests kann man aber schnell herausbekommen, welche Programme man besser nicht benutzen sollte.

Das Pascal + BASIC-System bietet mehrere Möglichkeiten, ins BASIC und danach zurück zum Pascal zu gelangen.

Wie man BASIC aufrufen kann:

ROMCALL -8192 als Einsprungadresse für das ROM-residente BASIC.

ROMCALL -155 oder -151 zum Eintritt in den Monitor.

LOADDOS zum Laden des DOS und Eintritt ins BASIC.

Wie man ins Pascal zurückkommt:

CALL 7 vom BASIC

CALL 1016 vom BASIC

"Ctrl-Y[RET]" vom Monitor

"7G[RET]" oder "3F8G[RET] vom Monitor

Funktionsweise

Die Idee ist ganz einfach: Das Pascal wird nach oben und das BASIC nach unten verschoben, damit sich die beiden Betriebssysteme nicht gegenseitig stören (s. Abbildung zur Speicheraufteilung). Tatsächlich teilen sich beide

Betriebssysteme den Speicherbereich über $D000, aber diese Teilung erfolgt störungsfrei, da nur die aktuell benötigte Speicherbank eingeschaltet wird. Das RAM der Language Card enthält das Pascal, während die ROMs auf der APPLE-Platine das BASIC enthalten. Zwischen beiden wird durch die Kontrollcodes $C088 und $C08A hin- und hergeschaltet (s. die Beschreibung im *APPLE Language System Installation and Operating Manual*).

Speicheraufteilung unter PBMENU

BASICSTUFF - Ohne BASICSTUFF ist PBMENU nicht funktionsfähig. Alle zur Hybridprogrammierung notwendigen Routinen wurden in der UNIT BASICSTUFF zusammengefaßt, die jedem Pascal-Programm mit der Deklaration USES BASICSTUFF zur Verfügung steht, nachdem sie in die SYSTEM.LIBRARY eingefügt worden ist. Hier eine kurze Beschreibung der bisher noch nicht erwähnten BASICSTUFF-Routinen:

MOVEHEAP gestattet es Ihnen, Ihr Pascal + BASIC-System den jeweiligen Erfordernissen der Speicheraufteilung anzupassen. Wenn Sie zum Beispiel ein langes Pascal-Programm haben, das ein kurzes BASIC- oder Maschinenprogramm aufruft, sollte der Heap nur ein wenig verschoben werden. Wenn Sie andererseits ein kleines Pascal-Programm verwenden, das ein langes BASIC-Programm benutzt, so ist es ggf. erforderlich, den Heap stark nach oben zu verschieben. Natürlich muß der Heap vor dem Rücksprung in den Command Level des Pascal-Betriebssystems wieder auf seine normale Startposition $C00 (3072) zurückgesetzt werden.

Eine noch raffiniertere Technik besteht darin, den Heap nur ein wenig nach oben zu verschieben, um ein langes Pascal-Segment und ein kurzes BASIC-Programm zu benutzen, dann den Heap weiter nach oben zu setzen, ein kleineres Pascal-Segment aufzurufen und ein langes BASIC-Programm auszuführen. Damit das Pascal geschützt ist, wird HIMEM: automatisch auf die Untergrenze des Heap gesetzt, falls das DOS (das unterhalb des Heaps liegen muß) nicht geschützt ist.

DISPLAY40 unterdrückt die Ausgabe der rechten Pascal-Textseite. Die meisten BASIC-Programme benutzen diesen Speicherbereich ($800 - BFF, die zweite Textseite).

DISPLAY80 wird benutzt, um die rechte Seite der Textdarstellung unter Pascal wiederherzustellen. Diese Prozedur wird normalerweise unmittelbar vor Termination eines Pascal(Hybrid)-Programms aufgerufen.

FPBASIC ist eine Funktion, die den Wert TRUE liefert, wenn APPLESOFT im ROM gelagert ist. Enthält das ROM Integer-BASIC, so liefert FPBASIC den Wert FALSE.

Unterschiede zwischen regulären und intrinsischen Units:

Reguläre Unit	Intrinsische Unit
Code der Unit wird zur Link-Zeit in das Hauptprogramm eingebunden.	Code der Unit wird zur Laufzeit in den Speicher geladen.
Hauptprogramm ist von der Unit unabhängig.	Hauptprogramm benötigt die Unit "on-line" in der SYSTEM.LIBRARY.
Unit verschwendet ggf. Speicherplatz, da ihr Code in jedes Hauptprogramm nochmals kopiert wird.	Unit spart Speicherplatz, da ihr Code nur einmal in der SYSTEM.LIBRARY gespeichert wird.

Mehr über Units

Das APPLE-Pascal unterstützt zwei Arten von Units. Jede Art hat ihre Vor- und Nachteile.

Wenn eine *reguläre Unit* mit dem Linker in ein Hauptprogramm eingebunden wird, wird bei diesem Vorgang der Code dieser Unit tatsächlich in das Hauptprogramm eingefügt, so daß die Unit ein Teil des Programms wird. Das ist ein Vorteil, da das Programm dadurch von der Original-Unit unabhängig wird. Wenn man diese Unit jedoch sehr häufig benutzt, wird ihr Code immer wieder in die Hauptprogramme eingebunden, was zu einer unnötigen Verschwendung von Diskettenkapazität führt.

Mit der *intrinsischen Unit* verhindert man dieses platzverschwendende Kopieren des Unitcodes, indem man ihn in die SYSTEM.LIBRARY einbaut. Wenn nun das Hauptprogramm gestartet wird, wird der Code der Unit automatisch in den Hauptspeicher geladen. Das Hauptprogramm funktioniert aber nicht, wenn die SYSTEM.LIBRARY (in welche die Unit eingebunden sein muß) zur Laufzeit nicht zur Verfügung steht.

Jede Unit besteht aus drei Teilen: *Interface, Implementation* und *Initialisierung*.

		Adresse
Monitor	P-Code	$F800
BASIC (ROM)	Interpreter	$E000
	BIOS (2.Bank)	$D000
	I/O-Adressen	$C000
	SYSCOM	$B000
	PBMENU	
	freier Speicher	
Pascal	Pascal Heap	$8001
BASIC	DOS (falls gealden)	$5600
entspricht einem 32K-Apple	frei verfügbarer Speicherplatz für BASIC-Programme	$800
	BASIC & DOS	

Abb. 1 Speicherplatzaufteilung von PBMENU

Der Interface-Teil der Unit ist als einziger für das Hauptprogramm "sichtbar". Die im Interface-Teil der Unit verwendeten Konstanten, Typen, Variablen, Funktionen und Prozeduren verhalten sich so, als wären sie Bestandteil des Hauptprogramms (d.h. sie sind global).

Der Implementation-Teil ist nur der Unit bekannt und kann daher vom Hauptprogramm nicht benutzt werden.

Der Initialisierungsteil wird beim Laden der Unit ausgeführt, bevor das Hauptprogramm gestartet wird.

Das Kopieren von DOS auf Pascal-Disketten

1) Kopieren Sie Ihre DOS 3.3-System-Master-Diskette mit einem Diskettenkopierprogramm (d.h. benutzen Sie nicht den INIT-Befehl und das FID-Programm).
2) Löschen Sie die nicht im ROM befindliche BASIC-Version von der gerade kopierten Diskette.

   Dadurch wird verhindert, daß das Pascal während des Bootens überschrieben wird.
3) Konvertieren Sie mit Hilfe des Pascal-Programms DOS32K die DOS 3.3-Master-Diskette in eine 32k-Disk.
4) Starten Sie PBMENU (achten Sie darauf, daß Sie mindestens 22 zusammenhängende, freie Diskettenblocks zur Verfügung haben).
5) Rufen Sie das BASIC mit ROM CALL -8192 auf.
6) Legen Sie die 32k-Master-Diskette in das Bootlaufwerk ein.
7) Geben Sie "PR #6«Ret»" ein, um die Diskette zu booten und das DOS in den Hauptspeicher zu laden.
8) Geben Sie "CALL 7" ein, damit Sie wieder ins Pascal zurückkommen.
9) Legen Sie wieder Ihre Pascal-Diskette in das Bootlaufwerk ein.
10) Benutzen Sie den BSAVE-Befehl mit folgenden Parametern:

```
DATEINAME = "DOS.3.3"
STARTADRESSE = 22016
LÄNGE = 10751
```

11) Re-initialisieren Sie das System durch Drücken der RESET-Taste.

Nunmehr sollte sich DOS 3.3 auf Ihrer Pascal-Diskette befinden.

Wenn Sie zwei Laufwerke haben, können Sie eines mit einer Pascal- und das andere mit einer BASIC-Diskette betreiben. Natürlich dürfen Sie den CATALOG-Befehl nicht auf eine Pascal-Diskette anwenden. Befindet sich

die BASIC-Diskette in Laufwerk Nr.2, so dürfen Sie nicht vergessen, "CATALOG,D2" einzugeben.

Ein paar Anmerkungen zur Hybridprogrammierung

Bei fast allen Hybridprogrammen müssen folgende Operationen vorgenommen werden, um Überschneidungen zwischen den beiden Betriebssystemen zu vermeiden:

1) Im allgemeinen verschiebt man zunächst den Heap nach oben, um Platz für Maschinen- oder BASIC-Programme zu schaffen. Man kann auch den Heap verschieben, um Pascal Code Patches zu verbergen, aber dafür gibt es einfachere und bessere Methoden, die ich bald veröffentlichen möchte.
2) Es ist wichtig, daß nur 40 Textspalten auf dem Bildschirm ausgegeben werden, wenn die Hauptspeicheradressen $800 bis $BFF (zweite Textseite) benutzt werden sollen.
3) Schließlich müssen Sie, nachdem das Hybridprogramm gelaufen ist, den Heap wieder auf seine alte Adresse setzen, sonst kann es passieren, daß Sie für Programme wie den SYSTEM.EDITOR o.ä. nicht mehr genug Speicherplatz zur Verfügung haben.

Das Programm PBMENU (Listing Nr.1) ist ein gutes Hybridbeispiel, nach dessen Muster Sie Ihre eigenen Programme schreiben können. Ein weiteres kurzes Beispiel finden Sie in Listing Nr.5. Das PROGRAM HYBRID demonstriert, wie einfach Hybridprogrammierung sein kann, wenn die UNIT BASICSTUFF erst einmal in der Bibliothek steht. Das Programm beginnt in Pascal, führt ein BASIC-Programm namens DEMO aus, startet dann mit BRUN eine Maschinenroutine mit dem Namen MUSIC, springt ins Pascal zurück, ruft mit ROMCALL eine Monitorroutine auf und kehrt schließlich wieder zu Pascal zurück - ein echtes Hybridprogramm. Bevor man ein Programm wie HYBRID ausführen kann, müssen die Maschinenroutinen und das BASIC mit einem Programm wie PBMENU auf einer Pascal-Diskette gespeichert werden. Nebenbei bemerkt müssen die WRITE-Anweisungen in HYBRID eingeklammert werden, damit sich das Programm nicht aufhängt.

Die Benutzung des Programms

Wenn Sie das alte Pascal (II.1) verwenden wollen, müssen Sie zwei Adressen (MAXCOL und SCR2) wie in Listing Nr.3 gezeigt ändern. Wenn Sie ein ex-

ternes Terminal oder eine 80-Zeichen-Karte verwenden, verändert sich das Pascal-System geringfügig, so daß gewisse Modifikationen notwendig werden (z.B. sind die Bildschirm-Kontrollcodes anders, der Heap beginnt bei $800 anstelle von $C00 usw.).

Es gibt zwei Methoden, das Pascal + BASIC-System zu installieren. Bei der ersten benutzt man BASICSTUFF in Form einer regulären Unit, was sich besonders zur Fehlersuche eignet. Bei der zweiten Methode wird BASICSTUFF als intrinsische Unit in die SYSTEM.LIBRARY eingebunden. Diese Methode eignet sich besser für die normale Hybridprogrammierung.

Methode Nr.1:

Zu Testzwecken kompilieren Sie die UNIT BASICSTUFF als Teil des Programms PBMENU. Das geschieht ganz einfach dadurch, daß man die gesamte Unit hinter den Programmanfang kopiert. Dabei müssen nur zwei kleine Änderungen vorgenommen werden: Zunächst muß man die Unit in eine reguläre Unit verwandeln, indem man den Text "INTRINSIC CODE 25 CODE 26" aus dem Kopf der Unit löscht. Als zweites muß der Punkt (.) am Ende der Unit in ein Semikolon (;) umgewandelt werden, damit der Compiler nicht schon an dieser Stelle abbricht, bevor er das Hauptprogramm erreicht. Wenn diese Änderungen eingetragen sind, können Sie das Hauptprogramm und die Unit zusammen kompilieren und mit den externen Prozeduren (PBPROC) verbinden. Wenn dann alle Routinen aus BASICSTUFF getestet sind und wunschgemäß laufen, kann die Unit wieder in eine INSTRINSIC UNIT zurückverwandelt, separat kompiliert, mit den Maschinenroutinen verbunden und in der SYSTEM.LIBRARY installiert werden.

Methode Nr.2:

Vielleicht wollen Sie BASICSTUFF gleich in die SYSTEM.LIBRARY einbauen. In diesem Fall muß die in Listing Nr.2 abgedruckte UNIT BASICSTUFF kompiliert und mittels des SYSTEM.LINKER mit den assemblierten Prozeduren aus Listing Nr.3 verbunden werden. Diese so kompilierte und verschränkte Unit muß dann mit dem Hilfsprogramm APPLE3: LIBRARY in der SYSTEM.LIBRARY installiert werden. Danach kann man jedes Hybridprogramm (wie z.B. PBMENU) von der Unit getrennt kompilieren und benutzen, um die faszinierende Welt des Pascal + BASIC-Systems zu erforschen. Weitere Einzelheiten über die Benutzung des Compilers, Assemblers, Linkers oder der Bibliothek finden Sie in Ihren Pascal-Handbüchern.

Schlußbemerkung

Es dauert vielleicht ein Weilchen, bis das Pascal + BASIC-System installiert ist, wenn das System aber läuft, haben Sie nicht nur eine Menge über Ihren Computer gelernt, sondern Ihnen stehen dann auch die außergewöhnlichen Möglichkeiten eines Hybridsystems zu Verfügung, mit dem Sie viele neue und interessante Phänomene entdecken können.

```
(*********************************************)
(* LISTING #1: PBMENU, EIN PASCAL+BASIC      *)
(*                        HYBRID-PROGRAMM    *)
(*                                           *)
(* VON RON DEGROAT   3/81                    *)
(*********************************************)

(*$S+,V-*)
PROGRAM PBMENU;

USES BASICSTUFF;

CONST BAS='.BAS';  (* Suffixe fuer Filenamen *)
      BIN='.BIN';
      NOTHING='';

      STARTOFHEAP=3072; (* Mit 80 Col Karte : *)
                        (* Start bei $800     *)

VAR HOME,ERASEOL,BELL,CH     :CHAR;
    F                        :FILE;
    TESTNAME,FILENAME        :STRING[25];
    SUFFIX                   :STRING[5];
    ADDR,CODELEN,NUMBLKS,I   :INTEGER;
    DONE                     :BOOLEAN;

PROCEDURE DOIT(CHOICE:INTEGER); FORWARD;

PROCEDURE PROMPTAT(LINE:INTEGER; MESSAGE:STRING);
BEGIN
  GOTOXY(0,LINE);
  WRITE(MESSAGE,ERASEOL);
END; (*PROMPTAT*)


PROCEDURE IOERRCHK;
VAR IOERR:INTEGER;
BEGIN
 IOERR:=IORESULT;
 IF IOERR<>0 THEN BEGIN
  CLOSE(F);
  WRITELN(BELL);
  WRITELN('EA-FEHLER NR.',IOERR);
  IF IOERR=10 THEN
```

```
   WRITELN('KEIN FILE  ',TESTNAME);
  WRITE('WEITER MIT LEERTASTE');
  READ(CH); EXIT(DOIT);
 END;
END; (*IOERRCHK*)

PROCEDURE FILECHK;
BEGIN
 TESTNAME:=CONCAT(FILENAME,SUFFIX);
(*$I-*)
 RESET(F,TESTNAME); IOERRCHK;
(*$I+*)
 CLOSE(F);
END;

PROCEDURE GETFILENAME;
BEGIN
 PROMPTAT(0,'FILENAME ==> ');
 READLN(FILENAME);
END;

PROCEDURE GETADDR(PROMPT:STRING);
BEGIN
 PROMPTAT(2,PROMPT);
 READLN(ADDR);
END;

PROCEDURE BINLOAD;
BEGIN
  PROMPTAT(22,'ERFORDERT VOLLSTAENDIGEN FILE NAMEN');
  PROMPTAT(23,'INCLUSIVE SUFFIX (BAS,BIN,CODE)');
  GETFILENAME;
  SUFFIX:=NOTHING; FILECHK;
  GETADDR('LADEADRESSE ==> ');
  BLOAD(FILENAME,ADDR);
END; (*BINLOAD*)


PROCEDURE BINSAVE;
BEGIN
 GETFILENAME;
 GETADDR('STARTADR ==> ');
 PROMPTAT(4,'LAENGE IN BYTES ==> ');
 READLN(CODELEN);
 BSAVE(FILENAME,ADDR,CODELEN);
END; (*BINSAVE*)


PROCEDURE BASICSAVE;
BEGIN
 GETFILENAME;
 SAVE(FILENAME);
END; (*SAVE*)


PROCEDURE BASICRUN;
BEGIN
 SUFFIX:=BAS;
 GETFILENAME; FILECHK;
```

```
  RUN(FILENAME);
 END; (*RUN*)


 PROCEDURE BINRUN;
 BEGIN
   SUFFIX:=BIN;
   GETFILENAME; FILECHK;
   BRUN(FILENAME);
 END; (*BRUN*)


 PROCEDURE CALLIT;
 BEGIN
   GETADDR('AUFZURUFENDE ADR ==> ');
   CALL(ADDR);
 END; (*CALL*)


 PROCEDURE ROMCALLIT;
 BEGIN
   GETADDR('AUFZURUFENDE ROM ADR ==> ');
   ROMCALL(ADDR);
 END; (*ROMCALL*)
PROCEDURE DOS;
BEGIN
 FILENAME:='DOS.3.3'; SUFFIX:=BIN;
 FILECHK;
 WRITE('LADE DOS...');
 LOADDOS;
END;


PROCEDURE DOIT;
BEGIN
 CASE CHOICE OF
   0:BINRUN;
   1:BINLOAD;
   2:BINSAVE;
   3:BASICRUN;
   4:BASICSAVE;
   5:DOS;
   6:CALLIT;
   7:ROMCALLIT;
   8:DONE:=TRUE;
 END; (*CASE*)
END; (*DOIT*)


PROCEDURE MENUOPTIONS;

CONST NUMOPTIONS=9;
      STARTMENU=3;
      SP=' ';

TYPE OPTIONS=STRING[10];

VAR MENU:PACKED ARRAY[0..NUMOPTIONS]
           OF OPTIONS;
```

```
    NEXT,SELECTION:INTEGER;

    INVERSE,NORMAL,
    LEFT,RIGHT:CHAR;

PROCEDURE INITMENU;
BEGIN
  MENU[0]:='BRUN';
  MENU[1]:='BLOAD';
  MENU[2]:='BSAVE';
  MENU[3]:='RUN';
  MENU[4]:='SAVE';
  MENU[5]:='LOAD DOS';
  MENU[6]:='CALL';
  MENU[7]:='ROM CALL';
  MENU[8]:='QUIT';
  SELECTION:=0;
END; (*INITMENU*)

PROCEDURE DISPLAYMENU;
BEGIN
  WRITE(HOME);
  WRITE('PASCAL+BASIC: HAUPTMENUE');
  GOTOXY(0,STARTMENU);
  FOR I:=0 TO NUMOPTIONS-1 DO
    BEGIN
      WRITELN(SP:14,MENU[I]);
      WRITELN;
    END;
  PROMPTAT(22,'LINKS- UND RECHTSPFEILE ZUM BEWEGEN DES CURSORS');
  PROMPTAT(23,'MIT LEER- ODER RETURNTASTE WIRD OPTION AUSGEFUEHRT');
END;

PROCEDURE MAKESELECTION;
BEGIN
  REPEAT
   GOTOXY(0,STARTMENU+2*SELECTION);
   WRITE(SP:14,INVERSE,MENU[SELECTION]);
   READ(KEYBOARD,CH);
   IF (CH=LEFT) OR (CH=RIGHT) THEN
    BEGIN
     GOTOXY(0,STARTMENU+2*SELECTION);
     WRITELN(NORMAL,SP:14,MENU[SELECTION]);
     IF CH=RIGHT THEN NEXT:=1
      ELSE NEXT:=-1;
       NEXT:=NEXT+NUMOPTIONS;
     SELECTION:=(SELECTION+NEXT) MOD NUMOPTIONS;
    END
  UNTIL CH=SP;
  WRITE(NORMAL,HOME);
END;

PROCEDURE MENUOPTIONS;

CONST NUMOPTIONS=9;
      STARTMENU=3;
      SP=' ';

TYPE OPTIONS=STRING[10];
```

```
VAR MENU:PACKED ARRAY[0..NUMOPTIONS]
           OF OPTIONS;

    NEXT,SELECTION:INTEGER;

    INVERSE,NORMAL,
    LEFT,RIGHT:CHAR;

PROCEDURE INITMENU;
BEGIN
  MENU[0]:='BRUN';
  MENU[1]:='BLOAD';
  MENU[2]:='BSAVE';
  MENU[3]:='RUN';
  MENU[4]:='SAVE';
  MENU[5]:='LOAD DOS';
  MENU[6]:='CALL';
  MENU[7]:='ROM CALL';
  MENU[8]:='QUIT';
  SELECTION:=0;
END; (*INITMENU*)

PROCEDURE DISPLAYMENU;
BEGIN
  WRITE(HOME);
  WRITE('PASCAL+BASIC: HAUPTMENUE');
  GOTOXY(0,STARTMENU);
  FOR I:=0 TO NUMOPTIONS-1 DO
    BEGIN
      WRITELN(SP:14,MENU[I]);
      WRITELN;
    END;
  PROMPTAT(22,'LINKS- UND RECHTSPFEILE ZUM BEWEGEN DES CURSORS');
  PROMPTAT(23,'MIT LEER- ODER RETURNTASTE WIRD OPTION AUSGEFUEHRT');
END;

PROCEDURE MAKESELECTION;
BEGIN
  REPEAT
   GOTOXY(0,STARTMENU+2*SELECTION);
   WRITE(SP:14,INVERSE,MENU[SELECTION]);
   READ(KEYBOARD,CH);
   IF (CH=LEFT) OR (CH=RIGHT) THEN
    BEGIN
     GOTOXY(0,STARTMENU+2*SELECTION);
     WRITELN(NORMAL,SP:14,MENU[SELECTION]);
     IF CH=RIGHT THEN NEXT:=1
      ELSE NEXT:=-1;
       NEXT:=NEXT+NUMOPTIONS;
     SELECTION:=(SELECTION+NEXT) MOD NUMOPTIONS;
    END
  UNTIL CH=SP;
  WRITE(NORMAL,HOME);
END;

BEGIN (*MENUOPTIONS*)

 INVERSE:=CHR(18);      (*CTL-R*)
 NORMAL:=CHR(20);       (*CTL-T*)
```

```
 RIGHT:=CHR(21);        (*CTL-U, -> *)
 LEFT:=CHR(8);          (*CTL-H, <- *)
 DONE:=FALSE;

 INITMENU;

 REPEAT
   DISPLAYMENU;
   MAKESELECTION;
   DOIT(SELECTION);
 UNTIL DONE;
END; (*MENU*)


BEGIN (*MAIN PROGRAM*)

(*Die hier verwendeten Ctrl-Zeichen koennen bei*)
(*externen Terminals, Videokarten usw. ggf. abweichen*)

  BELL:=CHR(7);          (*CTL-G*)
  HOME:=CHR(12);         (*CTL-L*)
  ERASEOL:=CHR(29);      (*CTL-]*)

(*Nur linke Bildschirmhaelfte anzeigen*)

  DISPLAY40;

(*HEAP von $C00 nach $8001 verschieben*)

  MOVEHEAP(STARTOFHEAP,-32767);

  MENUOPTIONS;

  DISPLAY80;

  MOVEHEAP(-32767,STARTOFHEAP);

END.



(**********************************)
(* LISTING #2:     UNIT BASICSTUFF *)
(*                                *)
(* VON RON DEGROAT   3/81         *)
(**********************************)

(*$S+,V-*)
UNIT BASICSTUFF; INTRINSIC CODE 25 DATA 26;

(* Entfernt man "INTRINSIC CODE 25 DATA 26' *)
(* so wird BASICSTUFF zur regulaeren Unit.  *)

(* Diese Unit muss kompiliert, mit den in   *)
(* Listing Nr. 3 abgebildeten externen      *)
(* Prozeduren gelinkt und in die            *)
(* SYSTEM.LIBRARY eingebunden werden.       *)

INTERFACE
```

```
TYPE UNITSTR=STRING[25];

VAR BASZERPG:PACKED ARRAY[0..255] OF 0..255;
    UNITFID:FILE;

PROCEDURE LOADDOS;
PROCEDURE DOSRESET;
PROCEDURE DISPLAY40;
PROCEDURE DISPLAY80;
FUNCTION  FPBASIC:BOOLEAN;
PROCEDURE RUN(FILENAME:UNITSTR);
PROCEDURE SAVE(FILENAME:UNITSTR);
PROCEDURE BRUN(FILENAME:UNITSTR);
PROCEDURE CALL(ADDR:INTEGER);
PROCEDURE ROMCALL(ADDR:INTEGER);
PROCEDURE MOVEHEAP(OLDLOC,NEWLOC:INTEGER);
PROCEDURE BLOAD(FILENAMEE:UNITSTR; BEGINADDR:INTEGER);
PROCEDURE BSAVE(FILENAME:UNITSTR; BEGINADDR,CODELEN:INTEGER);


IMPLEMENTATION

CONST DOSLOADADDR=22016;
      DOSINIT=23940;      (* $5D84 *)

TYPE MAGIC=RECORD CASE BOOLEAN OF
            TRUE:(INTPART:INTEGER);
            FALSE:(PTRPART:^INTEGER);
           END;

     BYTE=0..255;
     PAGEOFMEM=PACKED ARRAY[0..255] OF BYTE;

VAR CHEAT,SOURCE,DEST:MAGIC;

    SUFFIX:STRING[5];

    NUMBLKS,I,IO,
    HIMEM,HIMEMADDR,
    ENDADDR,RUNADDR,LOADADDR:INTEGER;

    DOSLOADED:BOOLEAN;

    PBCODEINFO:PACKED RECORD
                 CODELENG,CODEADDR:INTEGER;
                 NAME:STRING[25];
                 STARTADDR:INTEGER;
                 BASICZERPG:PAGEOFMEM;
                 COMMENT:STRING;
                 FILLER:ARRAY[0..71] OF INTEGER;
               END;

PROCEDURE CALL; EXTERNAL;
PROCEDURE ROMCALL; EXTERNAL;
PROCEDURE DOSRESET; EXTERNAL;
PROCEDURE DISPLAY40; EXTERNAL;
PROCEDURE DISPLAY80; EXTERNAL;
PROCEDURE INITBASZERPG; EXTERNAL;
FUNCTION  FPBASIC; EXTERNAL;
```

```
PROCEDURE POKEWORD(ADDR,DATA:INTEGER);
BEGIN
  CHEAT.INTPART:=ADDR;
  CHEAT.PTRPART^:=DATA;
END; (*POKE*)

FUNCTION PEEKWORD(ADDR:INTEGER):INTEGER;
BEGIN
  CHEAT.INTPART:=ADDR;
  PEEKWORD:=CHEAT.PTRPART^;
END; (*PEEK*)

PROCEDURE MOVEHEAP;
(*HEAP beginnt normalerweise bei $C00 (3072) *)

CONST NP=90;          (* TOP OF HEAP *)

VAR HEAPINFO,HPSTOP,HEAPPTR,LEN,
    DISPLACEMENT:INTEGER;
    HEAP:^INTEGER;

BEGIN
  HEAPINFO:=PEEKWORD(98)+14;
  HPSTOP:=HEAPINFO+112;
  DISPLACEMENT:=NEWLOC-OLDLOC;
  HIMEM:=NEWLOC;

(*Heaplaenge ermitteln*)
  MARK(HEAP);    (*Heap so klein wie*)
  RELEASE(HEAP);   (*moeglich machen*)
  LEN:=PEEKWORD(NP)-OLDLOC;

(*Setze NP (TOP OF HEAP) fest*)
  POKEWORD(NP,PEEKWORD(NP)+DISPLACEMENT);

(*Zeiger aendern*)
  FOR I:=0 TO 2 DO  (*System Heap Zeiger*)
    BEGIN
      HEAPPTR:=PEEKWORD(HEAPINFO+I*2);
      POKEWORD(HEAPPTR,PEEKWORD(HEAPPTR)+DISPLACEMENT);
    END;

  HEAPPTR:=PEEKWORD(HEAPINFO); (*Zeiger verschieben*)
  REPEAT
    POKEWORD(HEAPINFO,HEAPPTR+DISPLACEMENT);
    HEAPINFO:=HEAPINFO+2;
    HEAPPTR:=PEEKWORD(HEAPINFO);
  UNTIL (HEAPINFO>HPSTOP);

(*Heap verschieben*)
  SOURCE.INTPART:=OLDLOC;
  DEST.INTPART:=NEWLOC;
  MOVELEFT(SOURCE.PTRPART^,DEST.PTRPART^,LEN);
END; (*MOVEHEAP*)

PROCEDURE SAVEFILE(FILENAME:UNITSTR;
                   BEGINADDR,CODELEN:INTEGER);
```

```
BEGIN
  CHEAT.INTPART:=BEGINADDR;
  PBCODEINFO.BASICZERPG:=BASZERPG;
  NUMBLKS:=(CODELEN + 511) DIV 512;
  REWRITE(UNITFID,FILENAME);
  IO:=BLOCKWRITE(UNITFID,PBCODEINFO,1,0);
  FOR I:=1 TO NUMBLKS DO BEGIN
   IO:=BLOCKWRITE(UNITFID,CHEAT.PTRPART^,1,I);
   CHEAT.INTPART:=CHEAT.INTPART+512;
  END;
  CLOSE(UNITFID,LOCK);
END; (*SAVEFILE*)

PROCEDURE BSAVE;
BEGIN
 WITH PBCODEINFOR DO BEGIN
  NAME:=CONCAT(FILENAME,'.BIN');
  STARTADDR:=BEGINADDR;
  CODELENG:=CODELEN;
  SAVEFILE(NAME,BEGINADDR,CODELEN);
 END;
END; (*BSAVE*)

PROCEDURE SAVE;
BEGIN
 WITH PBCODEINFO DO BEGIN
  NAME:=CONCAT(FILENAME,'.BAS');
  STARTADDR:=BASZERPG[104]*256+BASZERPG[103];
  ENDADDR:=BASZERPG[176]*256+BASZERPG[175];
  CODELENG:=ENDADDR-STARTADDR+1;
  SAVEFILE(NAME,STARTADDR,CODELENG);
 END;
END; (*SAVE*)

PROCEDURE BLOAD;
BEGIN
 CHEAT.INTPART:=BEGINADDR;
 I:=1;
 RESET(UNITFID,FILENAME);
 WHILE NOT EOF(UNITFID) DO
  BEGIN
   IO:=BLOCKREAD(UNITFID,CHEAT.PTRPART^,1,I);
   I:=I+1;
   CHEAT.INTPART:=CHEAT.INTPART+512;
  END;
 CLOSE(UNITFID,LOCK);
END; (*BLOAD*)

PROCEDURE SETHIMEM;
VAR HM:PACKED ARRAY[0..1] OF BYTE;
BEGIN
(*Integer in Bytearray konvertieren*)
 MOVELEFT(HIMEM,HM,2);
 BASZERPG[HIMEMADDR]:=HM[0];
 BASZERPG[HIMEMADDR+1]:=HM[1];
END;
```

```
PROCEDURE CONNECT(CSWL,CSWH,KSWL,KSWH:BYTE);
BEGIN
 BASZERPG[54]:=CSWL; BASZERPG[55]:=CSWH;
 BASZERPG[56]:=KSWL; BASZERPG[57]:=KSWH;
END;

PROCEDURE LOADFILE(FILENAME:UNITSTR);
BEGIN
 FILENAME:=CONCAT(FILENAME,SUFFIX);
(*PBCODEINFO holen*)
 RESET(UNITFID,FILENAME);
 IO:=BLOCKREAD(UNITFID,PBCODEINFO,1,0);
 CLOSE(UNITFID,LOCK);
 BASZERPG:=PBCODEINFO.BASICZERPG;
 LOADADDR:=PBCODEINFO.STARTADDR;
 IF DOSLOADED THEN CONNECT(189,94,129,94)
  ELSE CONNECT(240,253,27,253);
 IF SUFFIX='.BAS' THEN  (*ERR BYTE loeschen*)
  POKEWORD(LOADADDR-1,0);
 BLOAD(FILENAME,LOADADDR);
END; (*LOADFILE*)

PROCEDURE RUN;
BEGIN
  SUFFIX:='.BAS';
  LOADFILE(FILENAME);;
  SETHIMEM;
  ROMCALL(RUNADDR);
END; (*RUN*)

PROCEDURE BRUN;
BEGIN
  SUFFIX:='.BIN';
  LOADFILE(FILENAME);
  ROMCALL(LOADADDR);
END; (*BRUN*)

PROCEDURE LOADDOS;
BEGIN
  BLOAD('DOS.3.3.BIN',DOSLOADADDR);
  DOSLOADED:=TRUE;
  HIMEM:=DOSLOADADDR;      (* $5600 *)
  SETHIMEM;
(* DOS Vektoren von Seite drei holen *)
  SOURCE.INTPART:=24145; (* $5E51 *)
  DEST.INTPART:=976;     (* $3D0  *)
  MOVELEFT(SOURCE.PTRPART^,DEST.PTRPART^,47);
  DOSRESET; (*SOFT ENTRY Vektor aendern*)
  ROMCALL(DOSINIT);
END; (*DOS*)

PROCEDURE INITPBCODEINFO;
BEGIN
 WITH PBCODEINFO DO
  BEGIN
   CODELENG:=0;CODEADDR:=1;
   FILLCHAR(NAME,25,' ');
   STARTADDR:=0;
   FOR I:=0 TO 255 DO BASICZERPG[I]:=0;
```

```
   FOR I:=0 TO 71 DO FILLER[I]:=0;
   COMMENT:='P+B BY RON DEGROAT 2/81';
  END;
END; (*INITPB*)

BEGIN                    (*Hauptprogramm*)
 DOSLOADED:=FALSE;
 HIMEM:=3072; (* $C00, unteres Heapende*)
 INITPBCODEINFO; INITBASZERPG;
 IF FPBASIC THEN BEGIN
  RUNADDR:=-10906; (* $D566 *)
  HIMEMADDR:=115;  (* $73-74*)
 END
  ELSE BEGIN
   RUNADDR:=-4116; (* $EFEC *)
   HIMEMADDR:=76;  (* $4C-4D*)
  END;
END.  (*'.' in ';' umwandeln, wenn Unit und Hauptprogramm*)
      (*gemeinsam kompiliert werden sollen *)
```

```
;-------------------------------------------------------
;LISTING #3: PBPROC, VON BASICSTUFF VERWENDETE ROUTINEN
;
;VON RON DEGROAT       3/81
;-------------------------------------------------------

;Adresse vom Stack holen

        .MACRO POP   ;FORMAT: POP ADDR
        PLA
        STA %1
        PLA
        STA %1+1
        .ENDM

;Adresse auf Stack ablegen

        .MACRO PUSH  ;FORMAT: PUSH ADDR
        LDA %1+1
        PHA
        LDA %1
        PHA
        .ENDM

;Subroutine fuer ROM-Aufruf

        .MACRO RCALL ;FORMAT: RCALL ADDRESS
        STA 0C08A    ;ENABLE ROM
        JSR %1
        STA 0C088    ;RE-ENABLE RAM
        .ENDM

;Daten oder Routine laden (maximal 256 Bytes)

        .MACRO LOAD  ;FORMAT:
        LDY #00      ;LADE SOURCE,DEST,LEN
$1      LDA %1,Y
```

```
        STA %2,Y
        INY
        CPY #%3
        BNE $1
        .ENDM

        .PUBLIC BASZERPG

SETNORM .EQU 0FE84
INIT    .EQU 0FB2F
SETVID  .EQU 0FE93
SETKBD  .EQU 0FE89
FPCHRGET.EQU 0F10A
CHRGET  .EQU 0B0

MONVEC  .EQU 03F0
ZERPAG  .EQU 0000
STK     .EQU 0100
DBUF    .EQU 03A0   ;DISK INFO

MAXCOL  .EQU 0DB7C  ;0D9B9 FUER VER II.1
SCR2    .EQU 0DBA6  ;0D9CB FUER VER II.1


        .PROC INITBASZERPG

        .REF ZERTEMP,STKTEMP

        LOAD ZERPAG,ZERTEMP,000
        LOAD STKTEMP,ZERPAG,000 ;LOESCHEN
        RCALL INITZPG
        LOAD ZERPAG,BASZERPG,000
        LOAD ZERTEMP,ZERPAG,000
        RTS
INITZPG JSR SETNORM
        JSR INIT
        JSR SETVID
        JSR SETKBD
        LOAD FPCHRGET,CHRGET,01C
        RTS


        .FUNC FPBASIC


;Wenn Applesoft in ROM, dann FPBASIC = TRUE

        .REF RETPAS

        STA 0C08A  ;ROM AKTIVIEREN
        POP RETPAS ;PASCAL RUECKSPRUNGADRESSE RETTEN
        PLA        ;OFFSET AUSSER ACHT LASSEN
        PLA
        PLA
        PLA
        LDY #00
        LDA 0E000  ;INTEGER ODER FLOATING POINT BASIC
        CMP #04C
        BNE FALSE  ;SPRUNG, WENN INTEGER BASIC
        INY
```

```
FALSE   TYA         ;LSB = 1 WENN FP
        PHA         ;LSB = 0 WENN INT
        PHA
        PUSH RETPAS
        STA 0C088   ;RAM WAEHLEN
        RTS


        .PROC ROMCALL,1

;Zero Pages austauschen, Pascal retten, ROMs einschalten
;Danach JSR und bei Ruecksprung alten Zustand wiederherstellen

        .DEF ZERTEMP,STKTEMP,DEST,RETPAS
        .DEF SOFTEV,PWREDUP

        LOAD ZERPAG,ZERTEMP,000
        LOAD BASZERPG,ZERPAG,000
        LOAD DBUF,BUFTEMP,012

        LOAD MVEC,MONVEC,010
        LOAD RETJMP,007,003

        POP RETPAS    ;PASCAL RUECKSPRUNGADRESSE RETTEN
        POP DEST      ;ZIELADRESSE HOLEN
        RCALL ENTRY   ;INDIREKTER ZIELAUFRUF
        PUSH RETPAS   ;PASCAL RUECKSPRUNGADRESSE WIEDERHERSTELLEN

;Diskinformationen wiederherstellen, Pascal-Nullseite gegen
;BASIC-Nullseite austauschen und Kaltstartbyte loeschen

        LOAD BUFTEMP,DBUF,012
        LOAD ZERPAG,BASZERPG,000
        LOAD ZERTEMP,ZERPAG,000
        STY 03F4      ;KALTSTARTBYTE LOESCHEN
        RTS

;Stack retten und nach DEST springen

ENTRY   TSX
        STX SAVESP
        LOAD STK,STKTEMP,000
        JMP @DEST

RESTORE LOAD STKTEMP,STK,000
        LDX SAVESP
        TXS
        RTS

MVEC    .WORD 0FA59   ;BRK
SOFTEV  .WORD 0E003
PWREDUP .BYTE 045
        JMP 0FF59     ;RESET
RETJMP  JMP RESTORE
        JMP RESTORE
        .WORD 0FF65

DEST    .WORD 00
RETPAS  .WORD 00
```

```
SAVESP  .BYTE 00
ZERTEMP .BLOCK 0100
STKTEMP .BLOCK 0100
BUFTEMP .BLOCK 012


        .PROC DOSRESET

        .REF SOFTEV,PWREDUP

        LDA #0BF
        STA SOFTEV      ;RESET VEKTOR ZEIGT
        LDA #05D        ;AUF DOS
        STA SOFTEV+1
        LDA #0F8
        STA PWREDUP     ;KALTSTARTBYTE
        RTS

.PROC CALL,1

.REF RETPAS,DEST

POP RETPAS
POP DEST
PUSH RETPAS
JMP @DEST

.PROC DISPLAY40

LDA 0C083  ;RAM AUF 2. BANK
LDA 0C083  ;ZUM SCHREIBEN EINSCHALTEN
LDA #0EA   ;NOP
STA SCR2
STA SCR2+1
LDA #27    ;GROESSTE SPALTENNUMMER = 39
STA MAXCOL
LDA 0C088  ;1. BANK ANWAEHLEN UND GEGEN SCHREIBEN
           ;SCHUETZEN
RTS

.PROC DISPLAY80

LDA 0C083
LDA 0C083
LDA #0B0   ;BCS
STA SCR2
LDA #011
STA SCR2+1
LDA #04F   ;GROESSTE SPALTENNR. = 79
STA MAXCOL
LDA 0C088
RTS

.END
```

```
(*********************************)
(* LISTING #4: 32K DOS CONVERSION *)
(*                               *)
(* VON RON DEGROAT 3/81          *)
(*********************************)

PROGRAM DOS32K;

(*Dieses Programm modifiziert eine DOS 3.3 Diskette so, *)
(*dass sich das DOS wie bei einem 32K System verhaelt.  *)

VAR BLK:PACKED ARRAY[0..511] OF 0..255;
    BLT,BLN:INTEGER;
    S:FILE;

PROCEDURE WAITFORCR;
BEGIN
 WRITELN('WEITER MIT <RETURN>');
 READLN;
END;

BEGIN
 WRITELN; WRITELN
 ('LEGEN SIE EINE KOPIE DER DOS 3.3 MASTER DISK');
 WRITELN('IN DAS BOOTLAUFWERK (#4:) EIN');
 WAITFORCR;
 RESET(S,'#4:');
 BLN:=2;
 BLT:=BLOCKREAD(S,BLK,1,BLN);
 IF (BLK[260]=191) THEN
  BLK[260]:=127
   ELSE
    WRITELN('KONVERSION UNMOEGLICH');
 BLT:=BLOCKWRITE(S,BLK,1,BLN);
 WRITELN;WRITELN('WIEDER PASCAL DISK EINLEGEN');
 WAITFORCR;
END.
```

David N. Jones

CODEMAP

Haben Sie schon einmal vergeblich versucht, ein Pascal-Programm zu starten, nur um schließlich herauszufinden, daß die Datei nicht eingebunden war oder daß eine UNIT benutzt wurde, die nicht in Ihrer SYSTEM.LIBRARY stand? Oder daß das Programm überhaupt nicht ausführbar war? Das ist normalerweise kein Problem, wenn man das Programm selbst geschrieben hat und seine Funktionsweise kennt, aber bei den vielen Programmen, die den Benutzern von USCD-Pascal zur Verfügung stehen, können bisweilen überraschende Probleme auftreten. UCSD-Code-Dateien enthalten in Block Nr.0 ein Segment Dictionary, das eine ganze Reihe interessanter Informationen beinhaltet. An dieser Stelle setzt das CODEMAP-Programm an, welches das Segment Dictionary liest und bei möglichen Problemen mit den Codefiles helfen kann. CODEMAP gibt Auskunft über:

die Anzahl der Segmente eines segmentierten Programms
den Namen des Codefiles bzw. der Segmente
die Art des Segments
die Blockadresse und Länge des Segments in Bytes
die Slotnummer
die Blockadresse des Interface-Teils von Units
die Art des Codes (P-Code oder Maschinencode)
die Pascal-Version, mit der das Programm kompiliert wurde
die Slotnummern der aus der SYSTEM.LIBRARY benötigten Units

Einige der oben aufgeführten Möglichkeiten bedürfen möglicherweise einer näheren Erklärung. Zur Zeit gibt es zwei Versionen (A.d.Ü.: jetzt sind es schon drei) des APPLE-Pascal (1.0 und 1.1), und manchmal kann es von Interesse sein zu erfahren, mit welcher Version das Programm übersetzt wurde. Es kann wichtig sein, den Code-Typ zu wissen, wenn das Programm rechnerabhängige Routinen verwendet. Beachten Sie bitte, daß ein mit "6502" bezeichnetes Segment nicht notwendigerweise vollständig aus Assemblerroutinen besteht: Wenn in das Segment Maschinenroutinen eingebunden sind, wird es nämlich insgesamt mit der Bezeichnung 6502-Code versehen. Die Segmentart gibt an, ob das Programm eingebunden ist oder nicht,

eine UNIT oder ein DATA-Segment einer Unit darstellt. Jedes Segment beginnt bei einem neuen Block, und die Blockadresse bezeichnet den Startblock des betreffenden Segments relativ zum Anfang der Datei.

Die Slotnummer ist nur für die Systembibliothek interessant. Wenn Sie das CODEMAP-Programm auf die SYSTEM.LIBRARY anwenden, dann geben die Slotnummern an, welcher Slot sich auf die jeweilige Unit bezieht. CODEMAP gibt die Slotnummern der notwendigen intrinsischen Units an. Vergleicht man die Slotnummern, so erfährt man, welche Units benutzt werden. Man könnte CODEMAP etwas aufwendiger schreiben, so daß es die Segmentnamen aus der Systembibliothek liest und die Namen der notwendigen Units auflistet, anstatt deren Slotnummern.

Weitere Informationen über die Definitionen sowie einen Großteil der Angaben, die benötigt werden, um CODEMAP zu schreiben, finden Sie im *Operating System Reference Manual* auf den Seiten 266 bis 270. Einen Teil der benötigten Informationen liefert Ihnen auch das Programm LIBMAP von der APPLE3:-Diskette. LIBMAP läßt sich sowohl auf Bibliotheken als auch auf Codefiles anwenden. Listing Nr.1 enthält den Quelltext des CODEMAP-Programms, während Listing Nr.2 zeigt, wie z.B. der Output eines SYSTEM.COMPILERs aussehen kann.

Listing 2

------- Code Map fuer #9:system.compiler -------

Seg #	Name	Segment Art	Adr	Laen	Slot	Intrf	Typ	Version
0		gelinkt & ausfuehrbar	0	0	0	0	Undef.	K.A.
1	PASCALCO	gelinkt & ausfuehrbar	1	4486	1	0	P-Code (-)	1.1
2	COMPINIT	gelinkt & ausfuehrbar	10	3226	7	0	P-Code (-)	1.1
3	DECLARAT	gelinkt & ausfuehrbar	17	7574	8	0	P-Code (-)	1.1
4	BODYPART	gelinkt & ausfuehrbar	32	7208	9	0	P-Code (-)	1.1
5	ROUTINE	gelinkt & ausfuehrbar	47	2902	10	0	P-Code (-)	1.1
6	STATEMEN	gelinkt & ausfuehrbar	53	1598	11	0	P-Code (-)	1.1
7	CASESTAT	gelinkt & ausfuehrbar	57	436	12	0	P-Code (-)	1.1
8	FORSTATE	gelinkt & ausfuehrbar	58	512	13	0	P-Code (-)	1.1
9	BODY1	gelinkt & ausfuehrbar	59	326	14	0	P-Code (-)	1.1
10	BODY3	gelinkt & ausfuehrbar	60	858	15	0	P-Code (-)	1.1
11	WRITELIN	gelinkt & ausfuehrbar	62	818	16	0	P-Code (-)	1.1
12	UNITPART	gelinkt & ausfuehrbar	64	1600	17	0	P-Code (-)	1.1
13	COMPOPTI	gelinkt & ausfuehrbar	68	1060	18	0	P-Code (-)	1.1
14	NUMSTRIN	gelinkt & ausfuehrbar	71	864	19	0	P-Code (-)	1.1
15	FINISHUP	gelinkt & ausfuehrbar	73	672	20	0	P-Code (-)	1.1

Erforderliche intrinsische Segmente: Keins

```
(*                          Listing 1

                              CODEMAP
                    Ein Code File Hilfsprogramm
                                von
                           David N. Jones
                            14. Mai 1981

*)

(*$I-*)
PROGRAM CODEMAP;
CONST
   MAXSEG = 31;
   MAXSLOT= 15;
TYPE
   SEGRANGE   =0..MAXSEG;
   SEGDICRANGE=0..MAXSLOT;
   MTYPES=(UNDEF,PCODEMOST,PCODELEAST,PDP11,M8080,Z80,GA440,M6502,M6800,TI990);
   REVISIONS=(NONAPPLE,ONEZERO,ONEONE,FUTURE1,FUTURE2,FUTURE3,
              FUTURE4,FUTURE5);
   SEGSET = SET OF SEGRANGE;
   SEGDICREC = RECORD
                 DISKINFO: ARRAY[0..15] OF
                            RECORD
                              CODELENG, CODEADDR:INTEGER
                            END;
                 SEGNAME:  ARRAY[0..15] OF PACKED ARRAY[0..7] OF CHAR;
                 SEGKIND:  ARRAY[0..15] OF (LINKED,HOSTSEG,SEGPROC,UNITSEG,
                                            SEPRTSEG,UNLINKED_INTRINS,
                                            LINKED_INTRINS,DATASEG);
                 TEXTADDR: ARRAY[0..15] OF INTEGER;
                 SEGINFO:ARRAY[SEGDICRANGE] OF
                                            PACKED RECORD
                                              SEGNO:0..255;
                                              MACHTYPE:MTYPES;
                                              FILLER:0..1;
                                              MAJORREVISION:REVISIONS;
                                            END;
                 INTSEGSET:SEGSET;
                 FILLER2:ARRAY[0..109] OF INTEGER
               END  (* SEGDICREC *);
VAR
  SEGDIC:SEGDICREC;
  F:FILE;
  O:INTERACTIVE;
  SEGNOTREQ:BOOLEAN;

FUNCTION YESNO:BOOLEAN;
  VAR CH:CHAR;
BEGIN
  REPEAT
    WRITE(' J(a oder N(ein :');
    READ(CH);
    WRITELN
  UNTIL   CH IN['J','N','j','n'];
  YESNO:= CH IN['J','j'];
END;
```

```
PROCEDURE ERROR(MESSAGE:STRING);
BEGIN
  WRITELN;
  WRITELN('==>FEHLER ',MESSAGE);
  WRITE  ('   RETURN fuer Abbruch:');
  READLN;
  EXIT(CODEMAP)
END;

PROCEDURE INIT;
  VAR FCFNAME,OFNAME:STRING;
BEGIN
  CLOSE(F);
  WRITE('Name des Codefiles ==>');
  READLN(FCFNAME);
  RESET(F,FCFNAME);
  IF IORESULT<>0 THEN
    ERROR('beim Oeffnen des Codefiles');
  IF BLOCKREAD(F,SEGDIC,1,0) <> 1 THEN
    ERROR('beim Lesen des Segment Dictionaries');
  WRITELN('Name der Ausgabedatei ');
  WRITE('<cr> = CONSOLE: ===>');
  CLOSE(O);
  OFNAME :='';
  READLN(OFNAME);
  IF LENGTH(OFNAME) = 0 THEN OFNAME:='CONSOLE:';
  REWRITE(O,OFNAME);
  IF IORESULT<>0 THEN
    ERROR('beim Oeffnen des Ausgabefiles');
  PAGE(O);
  WRITELN(O,'------- Code Map fuer ',FCFNAME,' -------');
  WRITELN(O,' Seg #  Name     Segment Art          Adr
           ' Laen Slot Intrf  Typ     Version');
  WRITELN(O,'________________________________________',
           '_______________________________________');
END;

PROCEDURE MAP;
VAR I: INTEGER; J: SEGRANGE;

PROCEDURE MAP1;
BEGIN
 WITH SEGDIC DO
    BEGIN
       WRITE(O,' ',I:4,' ',SEGNAME[I]);
       CASE SEGKIND[I] OF
          LINKED           : WRITE(O,' gelinkt & ausfuehrbar     ');
          HOSTSEG          : WRITE(O,' ungelinktes Hauptprogramm');
          SEGPROC          : WRITE(O,' Segmentprozedur          ');
          UNITSEG          : WRITE(O,' Regulaere UNIT           ');
          SEPRTSEG         : WRITE(O,' Seperate Prozedur        ');
          UNLINKED_INTRINS : WRITE(O,' ungelinkte INTRINSIC Unit');
          LINKED_INTRINS   : WRITE(O,' Gelinkte INTRINSIC Unit  ');
          DATASEG          : WRITE(O,' Datensegment             ');
       END; (* OF CASE *)
       WRITE(O,' ',DISKINFO[I].CODEADDR:4,DISKINFO[I].CODELENG:6);
    END; (* WITH SEGDIC *)
END; (* MAP1 *)
```

```
PROCEDURE MAP2;
  BEGIN
    WITH SEGDIC DO
      BEGIN
        WRITE(0,SEGINFO[I].SEGNO:5,TEXTADDR[I]:5,'  ');
        CASE SEGINFO[I].MACHTYPE OF
          UNDEF      :WRITE(0,' Undef.    ');
          PCODEMOST  :WRITE(0,' P-Code (+)');
          PCODELEAST:WRITE(0,' P-Code (-)');
          PDP11      :WRITE(0,' PDP11     ');
          M8080      :WRITE(0,' 8080      ');
          Z80        :WRITE(0,' Z80       ');
          GA440      :WRITE(0,' GA440     ');
          M6502      :WRITE(0,' 6502      ');
          M6800      :WRITE(0,' 6800      ');
          TI990      :WRITE(0,' TI990     ');
        END; (* OF CASE *)
        CASE SEGINFO[I].MAJORREVISION OF
          NONAPPLE : WRITE(0,' K.A.');
          ONEZERO  : WRITE(0,'  1.0');
          ONEONE   : WRITE(0,'  1.1');
        END; (* OF CASE *)
      END; (* WITH SEGDIC *)
    END; (* MAP2 *)

BEGIN
  FOR I:= 0 TO MAXSLOT DO
    BEGIN
       MAP1;
       MAP2;
       WRITELN(0);
    END;
  WRITE(0,'Erforderliche intrinsische Segmente: ');
  SEGNOTREQ:=TRUE;
  WITH SEGDIC DO
    BEGIN
     FOR J := 0 TO MAXSEG DO
      BEGIN
        IF J IN INTSEGSET THEN
        BEGIN
          WRITE(0,J:3);
          SEGNOTREQ:=FALSE;
        END;
      END;
     IF SEGNOTREQ THEN WRITE(0,' Keins');
    END;
  WRITELN(0);
 END;

 BEGIN
   REPEAT
     INIT;
     MAP;
     WRITE('===> Weiteres Code File ? ');
   UNTIL NOT YESNO;
   CLOSE(0,LOCK);
 END.
```

Chris Wilson

Patch für nicht eingelegte Boot-Disk

Im allgemeinen ist das Problem des Disk-Swappings unter APPLE-Pascal 1.1 recht gut gelöst. Eine unglückliche Ausnahme ist der Fall, wenn ein Programm beendet wird und zu diesem Zeitpunkt die Boot-Disk nicht eingelegt ist. In diesem Fall gibt das Pascal einen entsprechenden Hinweis aus und fordert dazu auf, die Boot-Disk wieder einzulegen. Danach startet es das Boot-Laufwerk alle fünf Sekunden, um nachzusehen, ob die Diskette inzwischen eingelegt wurde. Diese Warteschleife ist äußerst lästig.

Das kleine Programm in diesem Kapitel modifiziert das Pascal dergestalt, daß nach der Meldung, die Boot-Disk einzulegen, die folgende Nachricht auf dem Bildschirm erscheint:

TYPE «SPACE» TO CONTINUE

Danach wartet das Programm geduldig darauf, daß man die Boot-Disk einlegt. Wenn Sie die Leertaste drücken, ohne daß die Boot-Disk eingelegt ist, wird die obige Meldung einfach wiederholt. Das geschieht solange, bis man die richtige Diskette eingelegt hat.

```
PROGRAM PATCHOS;

(* Block Nr. 15 von SYSTEM.PASCAL wird
   so gepatcht, dass die Warteschleife,
   die das Bootlaufwerk abfragt, durch
   einen Aufruf der SPACE WAIT Prozedur
   ersetzt wird.


    Alter Code          Neuer Code
-----------------*------------------
LDCI (C7)    0FA0 ! LOD (B6)     02 03
STL (CC)     01   ! CXP (CD)     00 16
SLDO1 (E8)        ! SLDC1 (01)
SLDL1 (D8)        ! SLDC0 (00)
LEQI (C8)         ! SLDC0 (00)
FJP (A1)     07   ! CBP (C2)     28
SLDO1 (E8)        ! STL (CC)     01
SLDC1 (01)        !
ADI (82)          !
SRO (AB)     01   !
UJP (B9)     F6   !

*)

TYPE
  HEXSTRING = STRING[2];

VAR
  F: FILE;
  B: PACKED ARRAY [0..40,0..511] OF 0..255;
  CH: CHAR;
  I: INTEGER;


FUNCTION CONV(HEX: CHAR): INTEGER;

BEGIN
IF HEX IN ['A'..'F'] THEN
  CONV := 10+(ORD(HEX)-ORD('A'))
ELSE IF HEX IN ['0'..'9'] THEN
  CONV := ORD(HEX)-ORD('0')
ELSE
  CONV := 0;
END;

PROCEDURE FIX(BLK, BYTE: INTEGER; OLD, NEW: HEXSTRING);

VAR
  IOLD,
  INEW: INTEGER;
```

```
BEGIN
IOLD := CONV(OLD[2])+(CONV(OLD[1])*16);
INEW := CONV(NEW[2])+(CONV(NEW[1])*16);
IF B[BLK,BYTE] = IOLD THEN
  B[BLK,BYTE] := INEW
ELSE
  BEGIN
  WRITELN('Byte bei Block ',BLK,' Offset ',BYTE);
  WRITELN('ist ',B[BLK,BYTE],', sollte sein ',IOLD);
  EXIT(PATCHOS);
  END;
END;

BEGIN
PAGE(OUTPUT);
WRITELN('Patchos (15-Feb-81)');
WRITELN;
WRITELN('Copyright (c) 1982 Chris Wilson');
WRITELN;
WRITELN('Legen Sie eine Diskette mit dem Original SYSTEM.PASCAL');
WRITELN('in Laufwerk 1 ein');
WRITE('Weiter mit Leertaste');
READ(KEYBOARD,CH);
WRITELN;
RESET(F,'#4:SYSTEM.PASCAL');
I := BLOCKREAD(F,B,41);
CLOSE(F);
FIX(15,369,'C7','B6');
FIX(15,370,'A0','02');
FIX(15,371,'0F','03');
FIX(15,372,'CC','CD');
FIX(15,373,'01','00');
FIX(15,374,'E8','16');
FIX(15,375,'D8','01');
FIX(15,376,'C8','00');
FIX(15,377,'A1','00');
FIX(15,378,'07','C2');
FIX(15,379,'E8','28');
FIX(15,380,'01','CC');
FIX(15,381,'82','01');
FIX(15,382,'AB','D7');
FIX(15,383,'01','D7');
FIX(15,384,'B9','D7');
FIX(15,385,'F6','D7');
REPEAT
  PAGE(OUTPUT);
  WRITELN('Legen Sie die zu modifizierende Diskette in Laufwerk 1 ein');
  WRITE('Weiter mit Leertaste');
  READ(KEYBOARD,CH);
  WRITELN;
  RESET(F,'#4:SYSTEM.PASCAL');
  I := BLOCKWRITE(F,B,41);
  CLOSE(F,LOCK);
  WRITE('Noch eine Diskette aendern (j/n): ');
  READ(CH);
  WRITELN
UNTIL (CH <> 'N') AND (CH <> 'n');
END.
```

Dana Schwartz

HUFFIN

Dateitransfer: Pascal nach DOS

Nachdem man erkannt hatte, daß die RWTS-Routinen (Read/Write Track and Sector) des DOS 3.3 einen Zugriff auf alle Blocks einer Pascal-Diskette gestatten, wurde es zu einer relativ einfachen Aufgabe, die Kenntnisse über die Struktur der Pascal-Directory und -Textdateien anzuwenden, um diese auf der Diskette zu finden und in eine DOS-Textdatei zu konvertieren. Das vorliegende MUFFIN-ähnliche Programm HUFFIN, das in einer problemorientierten Sprache geschrieben wurde, ist die Implementierung dieses Konzeptes.

Zeile 100 setzt HIMEM: um 2100 Bytes herab, um Platz für die RWTS-Routine und die Puffer für die Pascal-Blocks zu reservieren. In den Zeilen 120 bis 190 werden diese RWTS-Routine und die zugehörigen IOB und DCT mit POKE-Befehlen in den Hauptspeicher geschrieben. Diese Datenstrukturen sind im *DOS 3.3-Handbuch* auf den Seiten 87 bis 89 beschrieben.

Der Bildschirmaufbau wird in den Zeilen 200 bis 230 vorgenommen; darauf folgt eine Frage an den Benutzer nach den Slots und Laufwerken von DOS- und Pascal-Diskette. Danach wird der Benutzer aufgefordert, die Disketten einzulegen. Beachten Sie, daß bei Verwendung eines Laufwerkes ein mehrmaliges Wechseln der Disketten notwendig ist und daß aus diesem Grunde die Pascal-Quelldiskette schreibgeschützt werden sollte, um eventuelle Fehler zu vermeiden (das gilt ganz besonders bei den ersten Konvertierversuchen).

In Zeile 290 wird der Benutzer nach dem Namen der Pascal-Textdatei gefragt, die kopiert werden soll. Dabei ist kein Diskettenname anzugeben (Beispiel: DJS.TEXT). Gibt man keinen Namen an, so wird ein abgekürztes Directory-Listing der Pascal-Diskette ausgegeben (GOSUB 5000), und der Benutzer wird erneut nach einem Namen gefragt. Wird ein Name eingegeben, so wird das Directory der Pascal-Diskette abgesucht (GOSUB 1000), und wenn der Name gefunden wird, Dateityp und -größe geprüft. In den Zeilen 310 bis 320 werden daraufhin die Puffer und die DOS-Textdatei initialisiert.

Die Konversion findet in den Zeilen 330 bis 340 statt, wobei Benutzer eines Laufwerkes nötigenfalls darauf aufmerksam gemacht werden, die Diskette zu wechseln. Die Textzeichen werden einfach zu einem Ausgabestring (A$) zusammengekettet, bis ein CR (ASCII-Wert 13) auftritt. Dann wird das so aufgebaute String in die DOS-Datei geschrieben. Kleinbuchstaben werden nicht in Großbuchstaben umgewandelt. Für die Zeichen DLE (ASCII-Wert 16) und NULL (ASCII-Wert 0) ist eine Sonderbehandlung erforderlich (für deren Verwendung vgl. Beschreibung des Textdatei-Formats auf S.266 des *Pascal Operating System Manuals*). Um den Benutzer zu unterhalten, während das APPLESOFT die Strings und den Speicher bearbeitet, wird auf dem Monitor ein Textfenster gezeigt, das es gestattet, die Pascal-Zeilen zu lesen, die auf die DOS-Diskette geschrieben werden.

In den Zeilen 450 bis 470 wird der Durchlauf beendet, indem die DOS-Datei geschlossen und der HIMEM:-Wert wiederhergestellt wird.

Die Unterroutine in Zeile 1000 liest das Pascal-Directory in den Hauptspeicher ein und sucht nach dem angegebenen Dateinamen (N$). Die Anfangs- und Endblocks (Top und Bottom) werden in den Variablen TP und BT gespeichert, der Dateityp in TY. Wurde die Datei nicht gefunden, so wird TY der Wert -1 zugewiesen. Die Directory-Struktur wurde von APPLE auf S.9 des *Washington APPLE Pie* vom Dezember 1980 veröffentlicht.

Das bei Zeile 2000 beginnende Unterprogramm liest zwei Pascal-Blöcke in den Hauptspeicher (BK und BK + 1). Die Unterroutine bei 3000 wird benutzt, um das zu jedem Block (BL) gehörende Spur/Sektor-Paar TR und S1/S2 zuzuordnen. (Beachten Sie, daß jeder Block aus zwei Sektoren *einer* Spur besteht.)

Die bei 4000 liegende Routine ist der BASIC-Zugriff auf die zuvor mit POKE in den Speicher geschriebene RWTS-Routine. Die 256 Bytes von Spur (TR) und Sektor (SE) werden in den Puffer geschrieben, dessen höherwertiger Adreßteil den Wert BF hat. Das Lo-Byte der Adresse wurde vorher gespeichert und ändert sich während der Ausführung nicht.

Das Unterprogramm bei 5000 liest das Pascal-Directory und gibt die Dateinamen auf dem Bildschirm aus. Darauf folgt bei 9000 eine allgemeine Fehlerbehandlungsroutine. Sie versucht, die Ausgabedatei zu schließen (CLOSE), falls sie vorhanden ist. Dadurch wird es möglich, durch CTRL-C einen partiellen Dateitransfer zu bewirken.

```
10  REM

         HUFFIN
   PASCAL/DOS CONVERTER

    BY DANA J. SCHWARTZ

    WASHINGTON APPLE PI

CALL -A.P.P.L.E. * OCT., 1981

100 HI =  PEEK (115) +  PEEK (116) * 256 - 2100: HIMEM: HI
110  ONERR  GOTO 9000
120  DEF  FN MOD(X) = (X / 256 -  INT (X / 256)) * 256
130 RWTS = HI:ER = HI + 17:IOB = HI + 18:DCT = HI + 36:BUFF = HI + 40
140 TK = IOB + 4:SC = IOB + 5:HB = IOB + 9:G$ =  CHR$ (7):PA = 0:DR = 0
150  POKE RWTS,169: POKE RWTS + 1, INT (IOB / 256): POKE RWTS + 2,160: POKE RWTS
 + 3, FN MOD(IOB): POKE RWTS + 4,32: POKE RWTS + 5,217: POKE RWTS + 6,3: POKE RW
TS + 7,176: POKE RWTS + 8,1: POKE RWTS + 9,96
160  POKE RWTS + 10,173: POKE RWTS + 11, FN MOD(IOB + 13): POKE RWTS + 12, INT (
IOB + 13) / 256: POKE RWTS + 13,141: POKE RWTS + 14, FN MOD(ER): POKE RWTS + 15,
 INT (ER / 256): POKE RWTS + 16,96
170  POKE IOB,1: POKE IOB + 3,0: POKE IOB + 6, FN MOD(DCT): POKE IOB + 7, INT (D
CT / 256): POKE IOB + 8, FN MOD(BUFF)
180  POKE IOB + 10,0: POKE IOB + 11,0: POKE IOB + 12,1: POKE IOB + 13,0: POKE IO
B + 14,0: POKE IOB + 15,96: POKE IOB + 16,1:
190  POKE DCT,0: POKE DCT + 1,1: POKE DCT + 2,239: POKE DCT + 3,216
200  HOME : VTAB 2: HTAB 16: PRINT "HUFFIN": VTAB 5: PRINT " PASCAL TO DOS TEXT
FILE CONVERSION"
210  VTAB 7: HTAB 10: PRINT "BY DANA J. SCHWARTZ": HTAB 10: PRINT "WASHINGTON AP
PLE PI": VTAB 10: INVERSE
230  NORMAL : VTAB 11: POKE 34,10: POKE 35,22: ON DR > 0 GOTO 270
240  ON I GOTO 250: PRINT "SOURCE DISK: ": HTAB 8: PRINT "SLOT: ";: GET I$: PRIN
T I$:SS =  VAL (I$): HTAB 8: PRINT "DRIVE: ";: GET I$: PRINT I$:SD =  VAL (I$)
245  PRINT : PRINT "TARGET DISK: ": HTAB 8: PRINT "SLOT: ";: GET I$: PRINT I$:TS
 =  VAL (I$): HTAB 8: PRINT "DRIVE: ";: GET I$: PRINT I$:TD =  VAL (I$)
250  ON SS < 1 OR SS > 7 GOTO 240: ON TS < 1 OR TS > 7 GOTO 240: ON SD < 1 OR SD
 > 2 GOTO 240: ON TD < 1 OR TD > 2 GOTO 240
260 SS = SS * 16: IF SS = TS * 16 AND SD = TD THEN DR = 1
270  POKE IOB + 1,SS: POKE IOB + 2,SD: IF DR = 1 THEN  PRINT  CHR$ (7): PRINT "I
NSERT WRITE PROTECTED PASCAL DISK": GOTO 290
280  ON J GOTO 300: PRINT : PRINT "WRITE PROTECT PASCAL SOURCE DISK AND   INSER
T IN SOURCE DRIVE": PRINT "INSERT DOS 3.3 TARGET DISK IN TARGET DR"
290  ON J GOTO 300: PRINT : PRINT "(HIT [C/R] FOR DIRECTORY)": PRINT "FILE NAME:
 ";: INPUT N$: IF  NOT  LEN (N$) THEN  GOSUB 5000: PRINT "FILENAME: ";:J = 1: IN
PUT N$: ON N$ = "" GOTO 470: GOTO 200
300  GOSUB 1000: IF TY <  > 3 OR BT - TP < 4 THEN  PRINT : INVERSE  PRINT G$"FI
LE EMPTY, NOT TEXT OR NOT FOUND": NORMAL :J = 0: GOTO 290
310  HOME :B1 =  INT (BUFF / 256):B2 = B1 + 1:B3 = B2 + 1:B4 = B3 + 1:BK = TP +
2: GOSUB 2000
320 A$ = "": PRINT  CHR$ (4)"OPEN"N$",S"TS",D"TD:D$ =  CHR$ (4): PRINT D$"MONO"
330  PRINT D$"WRITE"N$
340  FOR I = BUFF TO BUFF + 1023
350 C =  PEEK (I): IF C > 16 THEN A$ = A$ +  CHR$ (C): GOTO 410
360  IF C = 13 THEN  PRINT A$:A$ = "":Y =  FRE (0): GOTO 410
370  ON C <  > 16 GOTO 400
380 I = I + 1:SP =  PEEK (I): ON SP < 33 GOTO 410
```

```
390  FOR S = 1 TO SP - 32:A$ = A$ + " ": NEXT S: GOTO 410
400  IF C = 0 THEN I = BUFF + 1023
410  NEXT I
420 BK = BK + 2: ON BK = BT GOTO 450
430  IF DR = 1 THEN  PRINT D$"PR#0": HOME   PRINT  CHR$ (7)"INSERT PASCAL DISK A
ND HIT RETURN": GET I$: PRINT : HOME
440  GOSUB 2000: GOTO 330
450  TEXT   HOME
460  IF D$ =  CHR$ (4) THEN  PRINT D$"CLOSE": PRINT D$"NOMONO"
465  HOME : VTAB 4: PRINT "ANOTHER FILE ? ";: GET I$:I = 1: ON I$ = "Y" GOTO 200
470  HIMEM: HI + 2100: END : REM

1000  REM

*** FIND PASCAL FILE ***

1010 BF =  INT (BUFF / 256):TR = 0: FOR SE = 11 TO 4 STEP  - 1: GOSUB 4000:BF =
BF + 1: NEXT SE
1020 NU =  PEEK (BUFF + 16):PT = BUFF + 32:LN =  LEN (N$)
1030  ON  PEEK (PT) <  > LN GOTO 1110
1040  FOR J = 1 TO LN
1050  ON  PEEK (PT + J) <  >  ASC ( MID$ (N$,J,1)) GOTO 1110
1060  NEXT J
1070 TP =  PEEK (PT - 6) +  PEEK (PT - 5) * 256
1080 BT =  PEEK (PT - 4) +  PEEK (PT - 3) * 256
1090 TY =  PEEK (PT - 2)
1100  RETURN
1110 PT = PT + 26:NU = NU - 1: ON NU > 0 GOTO 1030
1120 TY =  - 1: RETURN
2000  REM
*** READ 2 PASCAL BLOCKS ***

2010 BL = BK: GOSUB 3000
2020 BF = B1:SE = S1: GOSUB 4000
2030 BF = B2:SE = S2: GOSUB 4000
2040 BL = BK + 1: GOSUB.3000
2050 BF = B3:SE = S1: GOSUB 4000
2060 BF = B4:SE = S2: GOSUB 4000
2070  IF DR = 1 THEN  PRINT  CHR$ (7)"INSERT DOS DISK AND HIT RETURN ": GET I$:
PRINT : HOME
2080  RETURN
3000  REM

*** BLK -> TR/SE ***

3010 TR =  INT (BL / 8):TMP = (BL / 8 - TR) * 8
3020 S2 = 2 * (7 - TMP):S1 = S2 + 1
3030  IF  NOT TMP THEN S1 = 0
3040  IF TMP = 7 THEN S2 = 15
3050  RETURN
4000  REM

*** CALL RWTS ***

4010  POKE TK,TR: POKE SC,SE: POKE HB,BF: POKE ER,0: CALL RWTS
4020  IF  NOT  PEEK (ER) THEN  RETURN
4030  IF D$ =  CHR$ (4) THEN  PRINT D$"PR#0": PRINT D$"NOMONO"
4040  TEXT   HOME : PRINT G$"RWTS DISK ERROR "; PEEK (ER): POP   POP   GOTO 9020
5000  REM
```

*** PASCAL DIRECTORY ***

```
5010  TEXT : HOME :BF =  INT (BUFF / 256):TR = 0: FOR SE = 11 TO 4 STEP  - 1: GO
SUB 4000:BF = BF + 1: NEXT SE
5020 V$ = "":NL = BUFF + 6: FOR I = 1 TO  PEEK (NL):V$ = V$ +  CHR$ ( PEEK (NL +
 I)): NEXT I: PRINT V$":":L$ = "":Y =  FRE (0)
5030 LN = 1:NF =  PEEK (BUFF + 16): IF  NOT NF THEN  PRINT G$;: FLASH : PRINT "[
NO FILES]": NORMAL  PRINT : PRINT "HIT [C/R] TO CONTINUE ": GET I$: PRINT : HOM
E : RETURN
5040  FOR I = 1 TO NF:ST = BUFF + I * 26 + 6:NL =  PEEK (ST): ON  NOT NL GOTO 50
60
5050  FOR J = 1 TO NL:L$ = L$ +  CHR$ ( PEEK (ST + J)): NEXT J: PRINT " "L$:L$ =
 "":Y =  FRE (0):LN = LN + 1
5060  IF LN > 20 OR I = NF THEN  PRINT   PRINT "HIT [C/R] TO CONTINUE ": GET I$:
 PRINT : PRINT V$": ";:LN = 1
5070  NEXT I: RETURN
9000  REM
```

ERROR HANDLER

```
9010  IF D$ =  CHR$ (4) THEN  PRINT D$"PR#0": PRINT D$"NOMONO"
9020  TEXT : PRINT   PRINT GS$"ERROR "; PEEK (222);" AT LINE "; PEEK (218) +  PE
EK (219) * 256
9030  POKE 216,0: IF (DR = 2 OR PA = 0) AND D$ =  CHR$ (4) THEN  PRINT D$"CLOSE"
9040  GOTO 470
9999  REM
```

MINOR MODS BY VAL J GOLDING

Dr. Wo/Washington Apple Pi

PUFFIN

Dateitransfer: DOS nach Pascal

Vor rund einem Jahr fragten mich Clubkameraden, die gerade angefangen hatten, Pascal zu lernen, ob es eine Möglichkeit gäbe, Dateien zwischen DOS und Pascal hin- und her zu kopieren. Da ich wußte, daß beide Systeme mit 16 Sektoren arbeiten, konnte ich die Frage ohne weiteres mit ja beantworten. Eines Tages machte ich mich daran, es zu beweisen.

Da ich ein unverbesserlicher, völlig einseitiger Pascal-Programmierer bin, erwies sich das Projekt für mich einerseits als etwas schwieriger, andererseits aber auch als etwas leichter, als ich zunächst angenommen hatte. Schwieriger, weil ich keine Ahnung vom DOS hatte und zunächst lernen mußte, wie DOS-Dateien verwaltet werden, und leichter, weil ich keine Veranlassung hatte (und auch heute noch nicht habe), Pascal-Dateien nach DOS zu kopieren und diesen Teil des Projektes daher einfach weggelassen habe.

Für jemanden, der nicht so "Pascal-borniert" ist, bleibt es natürlich nutzlos, nur den halben Satz von Programmen, die die Kommunikation zwischen zwei Betriebssystemen ermöglichen, zu haben. Zum Glück hat mich Dana Schwartz, einer meiner Clubkameraden, gerettet, indem er die andere Hälfte schrieb (s. das Kapitel HUFFIN).

Wozu braucht man PUFFIN?

Programme, mit denen man Daten zwischen verschiedenen Betriebssystemen hin- und herkopieren kann, sind natürlich schon deswegen interessant, weil man aus ihnen einiges über die Unterschiede zwischen den Betriebssystemen lernen kann. Darüberhinaus erweitern solche Programme auch die Anwendungsmöglichkeiten beider Systeme. Der Programmierer kann sich dann das für seine Zwecke am besten passende Betriebssystem aussuchen.

Aus meiner Sicht könnten ganz besonders die BASIC-Programmierer von diesen Konvertierprogrammen profitieren (wobei es mir schwer fällt, vorzustellen, was einen überhaupt noch veranlaßt, in BASIC zu program-

mieren). Mit diesen Programmen können Sie eine DOS-Datei, das ein BASIC-Programm in Textform enthält (zum LIST-Befehl vgl. S.76 des DOS-Handbuches) in eine Pascal-Datei konvertieren, das Programm mit dem Pascal-Editor lesen und verändern, es dann ins BASIC zurückkopieren und mit EXEC ausführen. Natürlich könnten Sie das ganze Programm auch gleich mit dem Pascal-Editor schreiben. Auf jeden Fall können Sie die beträchtlichen Möglichkeiten des Pascal-Editors zum Schreiben von BASIC-Programmen ausnutzen.

Umgekehrt können Pascal-Programmierer unter DOS erzeugte HI-RES-Grafiken in das Pascal-Betriebssystem übertragen. Das gleiche gilt für andere DOS-Dateien. Allerdings muß auf der Pascal-Seite noch einige zusätzliche Arbeit getan werden, zum Beispiel die Konversion von Dateien mit wahlfreiem Zugriff (Random Access Files) in Record-Strukturen, auf die dann Pascal-Programme zugreifen können. Da in Pascal die Wortgrenzen wichtig sind, kann das bedeuten, daß Sie in die Verlegenheit kommen könnten, für jede konvertierte Datei ein entsprechendes Programm schreiben zu müssen, es sei denn, sie würden ein äußerst flexibles Programm für diese zweite Konversionsstufe entwickeln. Die Notwendigkeit, die Wortgrenzen zu beachten und die von DOS einzeln benutzbaren Bytes zu zerlegen, ist in PUFFIN sehr anschaulich dargestellt.

Was macht PUFFIN?

PUFFIN ist ein Programm zum Transfer beliebiger DOS-Dateien in das Pascal-Betriebssystem. Es gibt die Möglichkeit, den Typ der Pascal-Datei, die erzeugt werden soll, unabhängig vom Typ der gewählten DOS-Datei frei zu bestimmen. Man kann zum Beispiel eine in Tokenform gespeicherte BASIC-Datei in eine Pascal-Datei transferieren, obwohl das Resultat augenscheinlich sinnlos ist. Sinnvollere Möglichkeiten sind bereits weiter oben im Text genannt worden. Die generierbaren Pascal-Dateitypen sind Text, Foto und Data.

PUFFIN bietet dem Benutzer ein aus vier Befehlen bestehendes Menü an: C(atalog, D(isplay, T(ransfer und Q(uit.

Catalog dient dazu, ein DOS-Directory zu lesen und auszugeben. Gibt man die Spezifikation eines Diskettenlaufwerkes nach den Pascal-Regeln ein (S.171 im *Language Manual* und S.276 im *Operating System Manual*), so sucht die Prozedur auf der angegebenen Diskette nach den DOS-Directory-Sektoren, baut aus ihnen die Directory-Informartionen auf und gibt das Ergebnis dieser Arbeit auf dem Monitor aus. Die dort angezeigten Informationen geben den Dateinamen an, den Dateityp, die Sektorlänge, die Adresse

der ersten Spur/Sektor-Liste sowie, ob die entsprechende Datei schreibgeschützt ist.

Display dient einfach dazu, die mit Catalog gesammelten Directory-Informationen noch einmal auf dem Bildschirm auszugeben. Sie werden als *aktuelles Directory* bezeichnet.

Transfer ist das Arbeitspferd des ganzen Programms. Diese Prozedur ermittelt den Namen der zu konvertierenden DOS-Datei, der Pascal-Zieldatei und den Typ der Pascal-Datei. Der Transfer findet nur statt, wenn sich das DOS-File im aktuellen Directory befindet und ein zulässiger Pascal-Dateiname angegeben wird.

Maßnahmen zur Vermeidung von E/A-Fehlern sind soweit ausgeschöpft, daß etwaiges Durcheinander, sei es bei Tastatur-Fehlern oder Disketten-E/A, sofort eingekreist wird. Das Programm gleitet Ihnen nicht aus den Händen, sondern läßt sich über Aufruf des Hauptmenüs zu jeder Zeit leicht kontrollieren.

Globale Deklarationen und das Hauptprogramm

Die globalen Deklarationen in PUFFIN definieren die Pascal-Repräsentation eines DOS-Directories und bilden damit die Grundlage der drei Befehle des Programms.

Beachten Sie, daß alle Strings durch Begriffe mit vorher definierten Konstanten definiert werden, die ihre Länge bestimmen, z.B.:

```
didleng = 30;
did = string[didleng];
```

Beachten Sie bitte auch, daß alle Arrays unter Verwendung von Konstanten deklariert sind, die ihre Dimensionierung bestimmen. Durch Typdeklarationen wird ihr Grundtyp bestimmt, und durch Untermengentypen wird der Indexbereich des Arrays festgelegt. Hier ein Beispiel:

```
TYPE
    blocksize = 512;
    blockrange = 0..blocksize;
    blockbuffer =
     PACKED ARRAY [1..blocksize] OF byte;

VAR
    block:blockbuffer;
    blockindex:blockrange;
```

Diese Deklarationen definieren, was ein "blockbuffer" ist und welchen Typ der Zeiger "blockindex" in die Variable "block" hat.

Durch Verwendung von Konstanten erhöht sich die Lesbarkeit des Programms. Gleichzeitig werden Änderungen erleichtert. Die Verwendung von Untermengen erhöht die Programmsicherheit, da eine Überschreitung der Bereichsgrenzen entweder zu Kompilations- oder Laufzeitfehlern führt - Pascal erlaubt keine Bereichsüberschreitungen. In komplizierteren Programmen ist das ein schwerwiegendes Plus, aber auch hier lohnt sich diszipliniertes Programmieren.

Die Deklaration des Datentyps "dosdirentry" wird den DOS-Freunden wahrscheinlich bekannt vorkommen. Beachten Sie, daß es zwei unterschiedliche Varianten von Directory-Einträgen gibt, die durch die Komponente "dfkind" unterschieden werden. Die eine Variante, bei der dfkind = volinfo ist, belegt nur den nullten Directory-Eintrag und enthält die Nummer der Unit, von der das Directory gelesen wurde, sowie die Anzahl der Directory-Einträge. Die andere Variante wird zur Beschreibung der eigentlichen Directory-Einträge verwendet und enthält natürlich auch die Komponente "dfkind", um anzugeben, um welchen DOS-Dateityp es sich jeweils handelt.

Das Hauptprogramm beginnt mit einer Initialisierung des Directories. Der nullte Eintrag wird in seiner "dfkind"-Komponente als "volinfo" deklariert, die Anzahl der Einträge wie auch die Gerätenummer für das DOS-Directory wird auf 0 gesetzt. Danach durchläuft das Programm eine Schleife, in der die Benutzeroptionen auf dem Monitor ausgegeben, gelesen und die gewünschten Befehle ausgeführt werden. Diese Schleife wird nur verlassen, wenn der Benutzer die Option "Quit" aufruft.

Catalog

Die Struktur eines DOS-Catalogs besteht aus einer linearen, verketteten Liste, in der jedes Element einen Diskettensektor belegt und aus zwei Komponenten besteht, nämlich einem Zeiger auf das nächste Element und einer Liste von bis zu sieben Directory-Einträgen. In Pascal kann dieser Zusammenhang wie folgt beschrieben werden:

```
dosnode    =RECORD
             nextnode:link;
             entries:ARRAY[1..7] OF dosdirentry;
             END;
```

wobei die Typen "link" und "dosdirentry" gültig in PUFFIN definiert worden sind.

Wenn der Aufbau eines DOS-Directories dieser Deklaration Byte für Byte entsprechen würde, so wäre es eine geradezu triviale Aufgabe, eine solche Liste abzuarbeiten, indem man die Directory-Sektoren sequentiell liest und nacheinander auf dem Bildschirm ausgibt. Unglücklicherweise ist das nicht der Fall, und es entsteht das zusätzliche Problem, daß man die in einem Directory-Sektor enthaltenen Informationen umformatieren muß. Obwohl das nicht schwierig ist, ist diese Aufgabe lästig, weil einzelne Bytes verschoben werden müssen. Abgesehen davon ist die Grundidee, eine verkettete Liste abzuarbeiten, immer noch das Kernproblem des Algorithmus.

Wie arbeiten wir die Liste dann ab und lesen das Directory? Nun, dazu brauchen wir nur zu wissen, wo die Liste beginnt, wann der letzte Sektor erreicht ist, und ob die gefundenen Einträge gültig oder gelöscht sind. Folgen wir den Angaben des DOS-Handbuches, so können wir davon ausgehen, daß ein Directory in Spur 17, Sektor 15 (dezimal) beginnt und daß der letzte Sektor erreicht ist, wenn der Zeiger für den nächsten Sektor auf Spur 0, Sektor 0 weist. Außerdem gehen wir davon aus, daß ein Directory-Eintrag inaktiv (d.h. gelöscht) ist, wenn der Zeiger für die Spur/Sektor-Liste dieses Eintrags (erstes Byte des Eintrages, d.h. Byte Nr.0) entweder 0 oder 255 ($FF) ist.

Catalog fragt zunächst nach der Nummer des Laufwerkes, in dem sich die zu lesende DOS-Diskette befindet. Gibt man Nr.0 ein, so wird die Kontrolle wieder an das Hauptmenü zurückgegeben; ansonsten wird eine Initialisierung der Variablen "dirlink" und "entrycount" durchgeführt. Beachten Sie, daß "dirlink" jetzt auf den ersten Sektor des DOS-Directories zeigt.

Es werden nun zwei geschachtelte WHILE-Schleifen aufgerufen, die das Traversieren des DOS-Directories und eine Übertragung der gefundenen Information in die Variable "dosdir" übernehmen.

Die äußere Schleife wird durch die Boole'sche Funktion "eodir" gesteuert, die prüft, ob noch ein weiterer Directory-Sektor gelesen werden muß. Die Funktion liefert nur dann den Wert FALSE, wenn die Variable "dirlink" auf Spur 0, Sektor 0 zeigt.

Die erste Aufgabe innerhalb der Schleife ist es, den Sektor, auf den "dirlink" zeigt, in die Variable "dirsector" einzulesen. Wenn dies fehlerfrei geschieht, initialisiert die Prozedur die Variable "sectorindex" und ruft die innere Schleife des Traversierungsteils auf.

Die innere Schleife wird durch "eodirsector", ebenfalls eine Boole'sche Funktion, kontrolliert. Diese Funktion sucht nach dem aktuellen Directory-Sektor und liefert genau dann den Wert FALSE, wenn sie einen aktiven Directory-Eintrag findet. Die Suche beginnt bei Eintrag Nr. "sectorindex + 1" und wird solange fortgesetzt, bis entweder ein aktiver Eintrag gefunden oder das Sektorende erreicht wird, je nachdem, welcher Fall zuerst eintritt. Das Sektorende ist erreicht, wenn alle sieben (= maxindex) potentiellen Einträge geprüft worden sind. Beachten Sie, daß durch die Initialisierung der Variablen "sectorindex" sichergestellt wird, daß die Suche

bei Eintrag Nr.1 beginnt.

Außer der Aktualisierung von "sectorindex" sorgt "eodirsector" auch für eine Aktualisierung der Variablen "entrybase". Letztere ist ein Zeiger auf einen Index von "dirsector", der das erste Informationsbyte eines Directory-Eintrages markiert.

An dieser Stelle der inneren Schleife zeigt "entrybase" auf den Anfang des nächsten in Pascal-Format zu konvertierenden Directory-Eintrages. Die Konversion erfolgt in drei Schritten. Als erstes wird die umzuformende Information (35 = entrylength Bytes) in die Variable "nextentry" kopiert, zweitens wird die Variable "entrycount" hochgezählt, und drittens wird die eigentliche Formatierung durch "filldirentry" ausgeführt. Sorgfältige Programmierer werden feststellen, daß sich die Variable "nextentry" und der Aufruf von "moveleft", durch den sie gefüllt wird, auch eliminieren lassen, wenn man "filldirentry" entsprechend ändert.

Nachdem der aktuelle Sektor abgearbeitet ist, aktualisiert die Prozedur "dirlink" und prüft die Bedingung für die äußere Schleife. Beim Verlassen dieser Schleife wird schließlich der nullte Eintrag des Directories aktualisiert und "displaydir" aufgerufen, um den Inhalt des Directories auf dem Monitor auszugeben.

Die "filldirentry"-Prozedur macht die eigentliche Schmutzarbeit beim Formatieren von DOS-Einträgen. Eine wichtige Aufgabe, die sie außerdem ausführt, ist die Konversion des DOS-Dateinamens in echte ASCII-Codes, indem sie das Hi-Bit der einzelnen Zeichen löscht. Zur Erhöhung der Programmsicherheit verwandelt sie nicht ausgebbare Zeichen in Blanks. Danach sucht sie das äußerste linke Leerzeichen und setzt dementsprechend die Länge des Dateinamens.

Displaydir

Die "displaydir"-Prozedur ist relativ simpel. Zuerst wird ein Vorlaufstück (Header) ausgegeben, und dann werden die Einträge untereinander auf den Bildschirm geschrieben. Es wurden die Voraussetzungen geschaffen, die Ausgabe zu stoppen, wenn der Bildschirm voll ist, und sogar die Möglichkeit, vorzeitig zum Hauptprogramm zurückzukehren. Eine Variable namens "cumsectors" enthält die Gesamtzahl der belegten Sektoren.

Die Prozeduren "displayheader" und "displayentry" sind global zu "displaydir" deklariert, da sie auch von "transfer" benutzt werden.

Die Directory-Ausgabe macht vom 80-Zeichen-Format des Pascal Gebrauch. Wer die Verwendung von Ctrl-A zur Seitenumschaltung vermeiden möchte, muß die Ausgabeprozeduren entsprechend modifizieren.

Transfer

Der Schlüssel zu einer DOS-Datei besteht in ihrer Spur/Sektor-Liste; wie der DOS-Catalog ist sie eine lineare, verkettete Liste. Die Elemente bestehen aus zwei Komponenten, einem Zeiger auf das nächste Listenelement (die Weiterführung der Spur/Sektor-Liste) und einer Liste von Zeigern, die auf die der Datei zugeordneten Diskettensektoren weisen, und zwar bis zu 122 (=maxlink) Stück in einem Listenelement. Die folgende Deklaration trägt dieser Datenstruktur Rechnung:

```
CONST
   maxlink = 122;
TYPE
   byterange = 0..255;
   link = PACKED RECORD
           tracknum:byterange;
           sectnum:Byterange;
         END;
   tslist = RECORD
           continuation:link;
           list:PACKED ARRAY [1..maxlink] OF link;
         END;
VAR
   currentlist:tslist;
```

Die Struktur der Spur/Sektor-Liste legt die Idee zu einem Traversierungsalgorithmus nahe, der dem in "Catalog" verwendeten ähnlich ist. Dazu gehören ebenfalls entsprechende Tests, wann das Listenende erreicht ist usw. Deren Funktionsweise haben wir ja bereits erläutert.

Die "transfer"-Prozedur beginnt mit zwei Schleifen, in denen ermittelt wird, wie die zu übertragende Datei heißt und wohin sie geschrieben werden soll. Zunächst wird gefragt, wie das zu konvertierende DOS-File heißt. Wird kein Name angegeben, dann gibt die Prozedur die Kontrolle an das Hauptprogramm zurück; ansonsten wird eine Schleife aufgerufen, die von der Boole'schen Funktion "searchdir" kontrolliert wird. Diese Funktion liefert dann und nur dann den Wert TRUE, wenn die angegebene Datei in dem aktuellen Directory steht. Wenn sie den Wert TRUE liefert, gibt sie auch gleichzeitig die Indexnummer des Eintrages.

Zur besseren Übersicht gibt die Prozedur danach die Directory-Informationen der gewählten Datei aus und initialisiert die Variable "nextnode" so, daß sie auf den ersten Sektor der Spur/Sektor-Liste der Datei weist.

An dieser Stelle ruft "transfer" die "getfiletype"-Prozedur auf, um herauszubekommen, welchen Pascal-Dateityp das DOS-File bekommen soll. Hier gibt es drei Möglichkeiten: text, foto und data.

Die "filetype"-Prozedur liefert ein String namens "suffix" und eine Variable mit dem Namen "filetype", deren Wertebereich aus "pasfilekinds" stammt. Das "suffix" wird an den Namen der Zieldatei angehängt, während "filetype" bestimmt, wie die Initialisierung und Formatierung der Pascal-Datei erfolgen soll.

Danach ermittelt die Prozedur den Namen der Pascal-Zieldatei und erreicht eine WHILE-Schleife, die von "openfile" kontrolliert wird. Diese Funktion versucht, die Zieldatei zu eröffnen, findet Fehler (wie z.B. unzulässige Dateinamen) und verhindert, daß das Programm bei dem Versuch, eine Datei zu öffnen, abstürzt. Achten Sie darauf, als Zieldiskette eine andere als die DOS-Diskette anzugeben.

Das Programm ist gegen Fehler der Art, die irreparable Dateiverluste nach sich ziehen können, nicht geschützt. Glauben Sie mir, ich habe meine Erfahrungen damit gemacht...

Die Prozedur initialisiert nun ihre Schlüsselvariablen. Die Übertragungspuffer "primpage" und "sparepage" werden mit Nullzeichen gefüllt und ihre entsprechenden Indexzeiger auf Null gesetzt. Das gleiche geschieht mit der Variablen "relblock", die die Anzahl der bisher geschriebenen Blöcke enthält. Wenn es sich bei der Datei um ein "fotofile" handelt, wird die Variable "fotoflag" auf TRUE gesetzt. Handelt es sich um eine Textdatei, werden zwei leere Blöcke als Textkopf der Datei auf die Diskette geschrieben. Diese beiden Blöcke werden vom Editor benutzt und für unsere Zwecke am besten mit Nullzeichen gefüllt.

Die Puffer "primpage" und "sparepage" sind jeweils zwei Blöcke lang und erleichtern damit die Übertragung von Textdateien in das Pascal-Betriebssystem. Diese beiden Puffer können leicht den Bedingungen für den Transfer in andere Dateitypen angepaßt werden.

Zum Schluß erreicht der Programmfluß die geschachtelten WHILE-Schleifen, die die eigentliche Informationsübertragung ausführen.

Die äußere Schleife wird durch die Boole'sche Funktion "eolist" kontrolliert, welche prüft, ob die Spur/Sektor-Liste fortgesetzt wird. Diese Funktion liefert genau dann den Wert TRUE, wenn "nextlink" auf Spur 0, Sektor 0 zeigt. Beachten Sie, daß durch die Initialisierung von "nextlist" sichergestellt wird, daß die Informationsübertragung an der richtigen Stelle beginnt.

Innerhalb dieser Schleife wird als erstes dasjenige Listenelement ermittelt, auf das "nextlink" zeigt. Dies geschieht durch die Funktion "get node", die genau dann den Wert FALSE liefert, wenn beim Lesen ein E/A-Fehler auftritt; ansonsten liefert sie den Wert TRUE und speichert den gelesenen Sektor in der Variablen "currentnode".

Nachdem das aktuelle Listenelement gelesen wurde, initialisiert "transfer" die Variable "linkindex" und erreicht die von der Funktion "eonode" kontrollierte innere WHILE-Schleife. Dieses Variablen-Funktions-Paar ähnelt sehr dem Paar "sectorindex" und "eodirsector", das von der Catalog-Prozedur verwendet wird. "Eonode" liefert nur dann den Wert FALSE, wenn sie einen aktiven Dateisektor findet; ist das der Fall, so liefert sie auch die Sektoradresse in "nextlink". Aktive Sektoren werden durch Spur/Sektor-Zeiger angezeigt, die nicht auf Spur 0, Sektor 0 weisen.

Bei der Abarbeitung der inneren Schleife wird "readtrksec" aufgerufen, die den Dateisektor, auf den "nextlink" zeigt, einliest und die Daten in "nextsector" speichert. Wenn alles klappt, wird die Information aus "nextsector" je nach Format der durch "filetype" bestimmten Datei in "primpage" umgespeichert. Wenn "primpage" nach dem Verlassen von "stuff" voll ist, werden die darin gespeicherten Daten mit "writeblocks" in die Zieldatei geschrieben. Die Kontrolle wird dann an die Abfrage für die innere Schleife übertragen. Beim Verlassen der inneren Schleife wird "nextlink" aktualisiert und die Kontrolle an die äußere Schleifenabfrage übergeben.

Writeblocks

Die Übertragung des "primpage"-Puffers wird durch die isolierte Prozedur "writeblocks" erledigt. Bei ihrem Aufruf sollte die Variable "pagepntr", die auf das zuletzt in "primpage" kopierte Zeichen zeigt, durch "blocksize" teilbar sein, und die Variable "blockcount" sollte die Anzahl der zu schreibenden Blöcke enthalten. Wenn der Aufruf der internen Funktion "blockwrite" einen anderen Wert als die Anzahl der zu schreibenden Blöcke liefert, wird "transfer" mit einem Aufruf von "abortxfer" abgebrochen; andernfalls wird "relblock" aktualisiert und die Kontrolle an "transfer" zurückgegeben.

Drei verschiedene Kopierarten

Die Feinheiten der Dateikonversion von DOS nach Pascal sind in der Prozedur "stuff" enthalten.

Die einfachste Konversionsart betrifft die Umformung einer DOS-Datei in ein Pascal-Datenfile. In diesem Fall wird die Datei Byte für Byte und Sektor für Sektor in das Pascal-File kopiert.

Beim Datentransfer in ein Fotofile wird davon ausgegangen, daß die Quelldatei eine Binärdatei ist, so daß die ersten vier Bytes der Datei die Start-

adresse und die Dateilänge in Bytes angeben (wahrscheinlich die Startadresse einer der beiden HI-RES-Grafikseiten und die Länge eines HI-RES-Bildes, d.h. 8192 Bytes). Diese beiden Angaben sind jedoch nur für das DOS wichtig. Entsprechend wurde die Unterprozedur "stuffoto" so ausgelegt, daß sie die ersten vier Bytes des ersten Datensektors ignoriert, aber alle Folgesektoren im Verhältnis 1:1 kopiert. Das Signal, die ersten vier Bytes nicht zu beachten, wird durch die Variable "fotoflag" gegeben. Beachten Sie, daß "stuffoto" als Eingabedatei nicht unbedingt ein HI-RES-Bild braucht, jedoch werden in jedem Fall die ersten vier Bytes weggelassen.

Die Prozedur "stuffoto" funktioniert bis auf die Behandlung des ersten Sektors wie folgt: Zuerst verschiebt sie den kompletten Inhalt von "sparepage" nach "primpage"; da immer nur ein Sektor auf einmal gelesen wird, wird auch nie mehr als ein Sektor nach "primpage" kopiert. Danach verschiebt sie so viele Bytes des aktuellen Sektors "s" wie möglich nach "primpage". Wenn das ein kompletter Sektor ist, wird "stuffoto" verlassen; ist es weniger, so wird der Differenzbetrag von "s" nach "sparepage" kopiert, und wir verlassen dann "stuffoto" mit einer vollgeschriebenen "primpage", die anschließend durch einen Aufruf von "writeblocks" wieder gelöscht wird.

Das Konvertieren in eine Textdatei ist etwas komplizierter. Man muß nämlich sicherstellen, daß die Zieldatei den Syntaxanforderungen von Pascal-Textdateien entspricht, wenn wir die Datei mit dem Pascal-Editor bearbeiten wollen. Die Kompatibilität mit dem SYSTEM.EDITOR wird durch die Prozedur "stufftext" sichergestellt.

Eine mit dem Editor kompatible Pascal-Textdatei kann - von oben nach unten - wie folgt definiert werden: Eine Textdatei besteht aus zwei Kopfblöcken zu je 512 Bytes, gefolgt von einer beliebigen Anzahl von Textseiten zu je 1024 Bytes. Jede Seite enthält eine Folge von Zeilen, und jede Zeile besteht aus einer Reihe von ASCII-Zeichen der folgenden Form:

(DLE Einrückung) (Zeichen) ... (Zeichen) (CR)

wobei DLE und CR die ASCII-Werte 16 bzw. 13 haben. Die "Einrückung" ist die Anzahl der führenden Leerzeichen einer Zeile plus 32, und "Zeichen" bezeichnet ein beliebiges ausgebbares ASCII-Zeichen. Zeilen dürfen über die Seitengrenzen nicht hinausragen, und auf die letzte Zeile einer Seite folgen so viele Nullzeichen (CHR(0)), wie notwendig sind, um die 1024 Bytes einer Seite aufzufüllen.

Beachten Sie, daß wegen dieser Bedingungen die - durchaus zu ertragende - Beschränkung einer Pascal-Zeile (nicht zu verwechseln mit einem umgangssprachlichen, deutschen Satz) auf nur 1024 Zeichen gilt.

Die "stufftext"-Prozedur transformiert ein DOS-File sektorweise in eine Pascal-Datei, wobei die Seiten- und Zeilengrenzen beachtet und alle gelese-

nen Zeichen in echte ASCII-Werte umgewandelt werden.

Die Prozedur beginnt mit der Konversion in ASCII-Werte, darauf werden alle nicht sichtbaren Steuerzeichen in Nullzeichen umgewandelt, schließlich werden die Nullzeichen eliminiert.

An dieser Stelle enthält der gelesene Sektor "s" "lengindex" zulässige Zeichen, die in den Puffer "primpage" übertragen werden können. Wenn man immer davon ausgehen könnte, daß keine gelesene Zeile mehr als 256 zulässige Zeichen enthält, dann könnte man "s" direkt nach "primpage" kopieren. Da wir das aber hier nicht tun, übertragen wir zunächst "s" nach "sparepage", die uns die Handhabung von Eingabezeilen mit bis zu 1024 Zeichen gestattet (was ja für Pascal das Maximum ist).

Die Prozedur erreicht nun eine WHILE-Schleife, die "sparepage" nach aufeinanderfolgenden Zeilen absucht und diese in die "primpage" überträgt. Zu diesem Zweck werden die Variablen "leadindex" und "lagindex" zusammen mit der Byte-orientierten Funktion "scan" benutzt. "lagindex" weist auf das Ende der zuletzt übertragenen Zeile und wird zu Beginn auf Null gesetzt. Die "scan"-Funktion dient in diesem Zusammenhang dazu, das nächste Carriage Return-Zeichen zu finden, wobei die Suche eine Stelle hinter "lagindex" begonnen wird. "leadindex" wird dazu benutzt, die von "scan" gefundene Stelle zu bezeichnen. Damit ist die Differenz "leadindex" - "lagindex" gleich der Anzahl der Zeichen einer Zeile, incl. des Carriage Return.

Die WHILE-Schleife wird beendet, wenn alle Zeichen aus "sparepage" abgesucht worden sind, oder wenn sicher ist, daß die nächste Zeile über die Seitengrenze hinausragen würde. Beim Verlassen dieser Schleife werden die Daten in "sparepage" um den entsprechenden Betrag nach links verschoben, um beim erneuten Aufruf von "stufftext" einen korrekten Start zu gewährleisten.

Der Brückenschlag

Die eigentliche Arbeit in diesem Programm wird durch eine Handvoll relativ maschinennaher Prozeduren und Funktionen erledigt. Die wichtigste von ihnen ist "readtrksec", die die Brücke von DOS nach Pascal darstellt und in der die unterschiedlichen Methoden der beiden Betriebssysteme, auf Disketten zuzugreifen, deutlich werden.

Das DOS behandelt natürlich Sektoren von 256 Bytes Länge als kleinste Diskettenspeichereinheit, und eine Diskette als eine Sammlung von jeweils 16 Sektoren, innerhalb von 35 Spuren. Das Pascal-Betriebssystem dagegen greift auf Blöcke von jeweils 512 Bytes Länge zu und teilt eine Diskette in 280 Blöcke ein. Beim Lesen eines DOS-Sektors von Pascal aus muß der ge-

samte Block, der den gewünschten Sektor enthält, gelesen und danach festgestellt werden, welche 256 Bytes daraus die richtigen sind.

Der DOS-Pascal-Tango

Haben Sie Ihr BASIC-Programm fertig? Ok, kopieren Sie es mit PUFFIN nach Pascal, übersetzen Sie es mit Ihrem Supercompiler (der natürlich in Pascal geschrieben ist) in 6502-Code und konvertieren Sie es dann mit Danas HUFFIN Programm wieder ins DOS-Format. Und ab geht die Post!

Was sagen Sie? Sie haben solch einen Compiler nicht?
Schade, hätte so schön sein können...!
Blaise, ade!!!

```
(*$S+*)
(*$V-*)
PROGRAM puffin;
CONST
 maxunit  =12; (* Groesste Pascal Unitnummer *)
 maxdir   =105; (* Groesstmoegliche Anzahl von DOS Directory Eintraegen *)
 maxlink  =122; (* Hoechstzahl fuer Eintraege in Spur/Sektor Liste *)
 didleng  =30; (* Groesste Laenge eines DOS Filenamens *)
 pidleng  =23; (* Maximale Laenge von Pascal Filenamen *)
 sidleng  = 5; (* Maximale Laenge von Pascal Filenamenssuffixen z.B. ".TEXT" *)
 sectsize =256; (* DOS Sektorgroesse *)
 blocksize=512;
 pagesize =1024; (* Textseitengroesse in Pascal *)
 maxbyte  =255;

 dirtrack    =17; (* Spurnummer, bei der das DOS Directory liegt *)
 firstdirsect=15; (* Erster Sektor eines DOS Directories *)

TYPE
 byterange =0..maxbyte;
 sectrange =0..sectsize;
 dirrange  =0..maxdir;
 linkrange =0..maxlink;
 unitrange =0..maxunit;
 blockrange=0..blocksize;
 pagerange =0..pagesize;

 sectbuffer =PACKED ARRAY[byterange] OF byterange;
 blockbuffer=PACKED ARRAY[1..blocksize] OF byterange;
 pagebuffer =PACKED ARRAY[1..pagesize] OF byterange;

 link=PACKED RECORD
 (* Wird zur Bezeichnung von Spur/Sektor Kombinationen benutzt *)
       tracknum:byterange;
       sectnum:byterange;
       END;
 tslist=(* Spur/Sektor Liste *)
        RECORD
         continuation:link;
         list:PACKED ARRAY[1..maxlink] OF link;
        END;

 did=STRING[didleng];
 pid=STRING[pidleng];
 sid=STRING[sidleng];

 dosfilekinds= (* DOS Filetypen *)
              (volinfo,unknown,dftext,dfinteger,applesoft,binary);
 pasfilekinds= (* Einige Pascal Filetypen *)
              (textfile,fotofile,untyped);

 (* Pascal Format der in einem DOS Directory enthaltenen Informationen *)
 dosdirentry=PACKED RECORD CASE dfkind:dosfilekinds OF
              volinfo: (* Hier das volume info *)
               (dunitnum:unitrange;
                dnumentries:dirrange);
              unknown,
              dftext,
              dfinteger,
```

```
  un:=stoi;
  IF NOT (un IN [0,4,5,9..12]) THEN writeln(chr(7));
 UNTIL un IN [0,4,5,9..12];
 unitnum:=un;
 get_unit_num:=(un<>0);
END;

PROCEDURE capitalize(VAR line:STRING);
CONST
 ordsmla=97;
 ordsmlz=122;
 shiftcase=32;
VAR
 index:0..maxbyte;
BEGIN
 FOR index:=1 TO length(line) DO
  IF line[index] IN [chr(ordsmla)..chr(ordsmlz)]
   THEN line[index]:=chr(ord(line[index])-shiftcase);
END;

FUNCTION getpasid(VAR name:pid):BOOLEAN;
BEGIN
 writeln;
 writeln('Geben Sie den Namen des Pascal-Zielfiles an.');
 writeln('Zum Abbrechen <RET> eingeben:');
 writeln;
 write('>>');
 readln(name);
 IF (length(name)=0) THEN getpasid:=FALSE
  ELSE BEGIN
   capitalize(name);
   getpasid:=TRUE;
  END;
END;

FUNCTION getdosid(VAR name:did):BOOLEAN;
BEGIN
 writeln;
 writeln('Name des zu transferierenden DOS-Files');
 writeln('Zum Abbrechen <RET> eingeben:');
 writeln;
 write('>>');
 readln(name);
 IF (length(name)=0) THEN getdosid:=FALSE
  ELSE BEGIN
   capitalize(name);
   getdosid:=TRUE;
  END;
END;

PROCEDURE getfiletype(VAR suffix:sid;VAR filetype:pasfilekinds);
BEGIN
 writeln;
 writeln('Transferieren als:');
 writeln;
 writeln('T)ext file, F)oto file, oder D)aten (binaer) file?');
 writeln;
 write('>> ');
 read(keyboard,ch);
```

```
 applesoft,
 binary:
  (file_tsl:link;  (* Adresse der Spur/Sektor Liste eines Files *)
   locked:BOOLEAN; (* Gibt an, ob File schreibgeschuetzt *)
   name:did;
   sectorcount:byterange);
   (* Anzahl der zugehoerigen Diskettensektoren *)
END;

 dosdirectory=ARRAY[dirrange] OF dosdirentry;
VAR
 dosdir:dosdirectory; (* Aktuelles DOS Directory *)
 unitnum:unitrange;
 ioerror:INTEGER;
 ch:CHAR;

 FUNCTION readtrksec(unitnum:unitrange;
                     trksec:link;VAR sb:sectbuffer;VAR ioerror:INTEGER):BOOLEAN;
 (* Liest Sektornummer 'trksec.sectnum' aus der Spur Nr. 'trksec.tracknum'
    von Unit Nr. 'unitnum' *)
 VAR
  block:blockbuffer;
  blocknum,offset:INTEGER;
 BEGIN
  WITH trksec DO
   BEGIN
    (* Berechne Halbblock, der dem gewuenschten Sektor entspricht *)
     IF (sectnum IN [0,15]) THEN blocknum:=sectnum
      ELSE blocknum:=15-sectnum;
     IF (odd(blocknum)) THEN offset:=256
      ELSE offset:=0;
     (* Berechne nun den Blocknummernoffset von Spur Null *)
     blocknum:=(blocknum DIV 2)+8*tracknum;
   END; (* WITH trksec DO *)
  (*$I-*)
  unitread(unitnum,block,sizeof(block),blocknum);
  (*$I+*)
  ioerror:=ioresult;
  IF NOT (ioerror=0) THEN readtrksec:=FALSE
   ELSE BEGIN
    (* Schreibe in Sektorpuffer *)
     moveleft(block[offset+1],sb,sizeof(sectbuffer));
     readtrksec:=TRUE;
   END; (* IF...THEN...ELSE *)
 END;

 FUNCTION writetrksec(unitnum:unitrange;
                     trksec:link;VAR sb:sectbuffer;VAR ioerror:INTEGER):BOOLEAN;
 VAR
  blocknum,offset:INTEGER;
  block:blockbuffer;
 BEGIN
  (* S. Kommentare zu 'readtrksec' *)
  WITH trksec DO
   BEGIN
    (* Berechne Halbblock, der dem gewuenschten Sektor entspricht *)
     IF (sectnum IN [0,15]) THEN blocknum:=sectnum
      ELSE blocknum:=15-sectnum;
     IF (odd(blocknum)) THEN offset:=256
      ELSE offset:=0;
```

```
    (* Berechne nun den Blocknummernoffset von Spur Null *)
    blocknum:=(blocknum DIV 2)+8*tracknum;
  END; (* WITH trksec DO *)
 (*$I-*)
 unitread(unitnum,block,sizeof(block),blocknum);
 (*$I+*)
 ioerror:=ioresult;
 IF NOT (ioerror=0) THEN writetrksec:=FALSE
  ELSE BEGIN
   moveleft(sb,block[offset+1],sizeof(sectbuffer));
   (*$I-*)
   unitwrite(unitnum,block,sizeof(block));
   (*$I+*)
   ioerror:=ioresult;
   writetrksec:=ioerror=0;
  END;
END;

FUNCTION searchdir(target:did;VAR index:dirrange):BOOLEAN;
VAR
 found:BOOLEAN;
BEGIN
 found:=FALSE;
 index:=dosdir[0].dnumentries;
 WHILE NOT (found OR (index=0)) DO
  BEGIN
   found:=target=dosdir[index].name;
   index:=index-1;
  END;
 IF found THEN index:=index+1;
 searchdir:=found;
END;
FUNCTION stoi:INTEGER;
VAR
 ch:CHAR;
 x:INTEGER;
BEGIN
 x:=0;
 read(ch);
 WHILE ch IN ['0'..'9'] DO
  BEGIN
   x:=10*x+(ord(ch)-ord('0'));
   read(ch);
  END;
 writeln;
 stoi:=x;
END;

FUNCTION get_unit_num(VAR unitnum:unitrange):BOOLEAN;
VAR
 un:INTEGER;
BEGIN
 REPEAT
  writeln;
  writeln('Geben Sie die Unitnummer [4,5,9..12] des Laufwerks an,');
  writeln('in dem sich die zu lesende DOS-Diskette befindet. Zum');
  writeln('Abbrechen "0" eingeben. ');
  writeln;
  write('>> ');
```

```
 WHILE NOT (ch IN ['t','f','d','T','F','D']) DO
  BEGIN write(chr(7));read(keyboard,ch); END;
 writeln(ch);
 CASE ch OF
  'T','t':BEGIN suffix:='.TEXT';filetype:=textfile; END;
  'F','f':BEGIN suffix:='.FOTO';filetype:=fotofile; END;
  'D','d':BEGIN suffix:='';filetype:=untyped; END;
 END;
END;

PROCEDURE printmenu;
CONST
 cleoln=29;
BEGIN
 gotoxy(0,0);
 write(chr(cleoln),'C)atalog, A)nzeige, T)ransfer, Q)uit?');
END;

PROCEDURE readcommand(VAR ch:CHAR);
BEGIN
 read(keyboard,ch);
 WHILE NOT(ch IN ['C','c','A','a','T','t','Q','q']) DO
  BEGIN
   write(chr(7));
   read(keyboard,ch);
  END;
 writeln;
END;

PROCEDURE displayentry(de:dosdirentry);
BEGIN
 WITH de DO
  BEGIN
   write(name,' ':(didleng-length(name)+1));
   CASE dfkind OF
    dftext:write('text':6);
    dfinteger:write('int':6);
    applesoft:write('soft':6);
    binary:write('bin ':6);
    unknown:write('unb ':6);
    END;
   IF locked THEN write('ja ':8)
    ELSE write('nein':8);
   write(sectorcount:9);
   writeln(filetsl.tracknum:6,'-',filetsl.sectnum:3);
  END;
END;

PROCEDURE displayheader;
BEGIN
 write('File Name');
 write('Typ ':((didleng-length('file name'))+7));
 write('Schr.g.':8);
 write('Sektoren':9);
 writeln('SSL link':10);
END;
```

```
PROCEDURE displaydir;
CONST
 cleos=11;
 esc=27;
 maxlines=21;
VAR
 cumsectors:INTEGER;
 count:dirrange;

BEGIN
 page(output);
 gotoxy(0,1);
 cumsectors:=0;
 IF dosdir[0].dnumentries=0 THEN writeln('Das aktuelle Directory ist leer!')
  ELSE BEGIN
    displayheader;
    FOR count:=1 TO dosdir[0].dnumentries DO
     BEGIN
      displayentry(dosdir[count]);
      cumsectors:=cumsectors+dosdir[count].sectorcount;
      IF (count MOD maxlines)=0 THEN
       BEGIN
        write('Weiter mit <RET>, Abbrechen mit <ESC> ');
        read(keyboard,ch);
        IF ch=chr(esc) THEN exit(displaydir)
         ELSE BEGIN gotoxy(0,2);write(chr(cleos)); END;
       END;
     END;
    write(dosdir[0].dnumentries,' Filea auf Diskette, ');
    write(cumsectors,' Sektoren belegt');
   END;
END;

PROCEDURE catalog;
CONST
 nextlink   = 1; (* Das Byte Nr. 1 relativ zum Directorysektor ist das Link
                    zum naechsten Directorysektor *)

 zerobase   =11; (* 1. Byte der Fileinformation in einem Directorysektor *)
 entrylength=35; (* DOS-Directoryeintraege belegen 35 Bytes *)
 mark      =maxbyte; (* Geloeschte Eintraege werden im (relativen) Byte Nr.0
                        "markiert" *)
 maxindex     7; (* Maximal sieben Directoryeintraege pro Sektor *)

 space= 32; (* ASCII Leerzeichen *)
 tilde=126; (* ASCII tilde *)
TYPE
 indexrange=0..maxindex;
 entrybuffer=PACKED ARRAY[1..entrylength] OF byterange;

VAR
 sectorindex:indexrange;
 entrybase:byterange;
 dir_link:link;
 dir_sector:sectbuffer;
 nextentry:entrybuffer;
 entrycount:dirrange;
```

```
FUNCTION eodir(dirlink:link):BOOLEAN;
BEGIN
 WITH dirlink DO
  eodir:=(sectnum=0) AND (tracknum=0);
END;



 PROCEDURE fill_dir_entry(VAR de:dosdirentry;VAR eb:entrybuffer);
 CONST
  linkoffset = 1; (* Relatives Byte Nr. null gibt die Lage seiner
                     Spur/Sektor Liste an *)
  kindoffset = 3; (* Relatives Byte Nr. zwei bezeichnet den Filetyp
                     des Eintrages *)
  nameoffset = 4; (* Mit relativem Byte Nr. drei beginnt der Filename *)
  countoffset=34; (* Relatives Byte Nr. 33 ist die Sektoranzahl
                     (MOD sectsize) der Datei *)
  lockbit   =128; (* In schreibgeschuetzten Dateien ist das hoechstwertige
                     Byte des Filetypenbytes gesetzt *)
 VAR
  j,kind:byterange;
  nonblank:0..didleng;
 BEGIN
  WITH de DO
   BEGIN
    filetsl.tracknum:=eb[linkoffset];
    filetsl.sectnum:=eb[linkoffset+1];
    kind:=eb[kindoffset];
    IF NOT ((kind MOD lockbit) IN [0,1,2,4]) THEN dfkind:=unknown
     ELSE CASE (kind MOD lockbit) OF
           0:dfkind:=dftext;
           1:dfkind:=dfinteger;
           2:dfkind:=applesoft;
           4:dfkind:=binary;
           END;
    IF ((kind DIV lockbit)=1) THEN locked:=TRUE
     ELSE locked:=FALSE;
    FOR j:=0 TO (didleng-1) DO
     BEGIN
      (* Hoechstwertiges Bit loeschen -> echter ASCII Wert *)
      eb[nameoffset+j]:=eb[nameoffset+j] MOD 128;
      (* Sonderzeichen loeschen *)
      IF NOT (eb[nameoffset+j] IN [space..tilde]) THEN eb[nameoffset+j]:=space;
     END;
    (* Letztes linkes Leerzeichen des Namensfeldes suchen *)
    nonblank:=-scan(-didleng,<>' ',eb[nameoffset+didleng-1]);

    (* Initialisierung der Laenge von 'name' *)
    (*$R-*)
    name[0]:=chr(didleng-nonblank);
    (*$R+*)
    (* Schliesslich Name einkopieren *)
    moveleft(eb[nameoffset],name[1],length(name));
    sectorcount:=eb[countoffset];
   END; (* WITH de DO *)
 END; (* filldirentry *)

FUNCTION eodirsector
  (VAR index:indexrange;
   VAR dirsector:sectbuffer;VAR entrybase:byterange):BOOLEAN;
```

```
VAR
 nofile:BOOLEAN;
BEGIN
 nofile:=TRUE;
 WHILE (nofile AND (index<maxindex)) DO
  BEGIN
   index:=index+1;
   entrybase:=zerobase+(index-1)*entrylength;
   nofile:=(dirsector[entrybase] IN [0,mark]);
  END;
 eodirsector:=nofile;
END;

BEGIN (* catalog *)
 page(output);
 IF NOT getunitnum(unitnum) THEN exit(catalog);
 WITH dir_link DO
  BEGIN
   tracknum:=dirtrack;
   sectnum:=firstdirsect;
  END;
 entrycount:=0;
 WHILE NOT eodir(dir_link) DO
  BEGIN
   IF NOT readtrksec(unitnum,dir_link,dir_sector,ioerror)
    THEN BEGIN writeln('E/A-Fehler ',ioerror,' beim Lesen des Directories');
               exit(catalog);
         END
    ELSE BEGIN
     sectorindex:=0;
     WHILE NOT eodirsector(sectorindex,dir_sector,entrybase) DO
     BEGIN
      moveleft(dir_sector[entrybase],nextentry,entrylength);
      entrycount:=entrycount+1;
      filldirentry(dosdir[entrycount],nextentry);
     END;
    END; (*IF...THEN...ELSE *)
   WITH dir_link DO
    BEGIN
     tracknum:=dir_sector[nextlink];
     sectnum:=dir_sector[nextlink+1];
    END;
  END;
 WITH dosdir[0] DO
  BEGIN
   dnumentries:=entrycount;
   dunitnum:=unitnum;
  END;
 displaydir;
END; (* catalog *)
```

```
(*$IDPTH2.1:TRANSFER.TEXT*)

BEGIN
 WITH dosdir[0] DO
  BEGIN dfkind:=volinfo; dnumentries:=0; dunitnum:=0; END;
 page(output);
 gotoxy(0,5);
 writeln('Willkommen bei PUFFIN!');
 REPEAT
  printmenu;
  readcommand(ch);
  CASE ch OF
   'c','C':catalog;
   'a','A':displaydir;
   't','T':transfer;
   END;
 UNTIL ch IN ['Q','q'];
END.
```

Ron DeGroat

Pascal Speicherdiagnose (PMU)

Das PMU-Programm (Pascal Memory Utility) wurde zur Unterstützung für den Benutzer entwickelt, das Apple Pascal System in seine Komponenten zu zerlegen und sich in der untersten Systemebene zurechtzufinden. Mit PMU kann man über eine große Anzahl von Benutzerbefehlen alle Teile des Pascal-Betriebssystems disassemblieren, untersuchen oder in beliebiger Weise modifizieren. Die auffälligste Charakteristik dieses Programms ist die Verwendung von Monitor-ROM-Subroutinen für den größten Teil der PMU-Funktionen, insbesondere für E/A-Operationen. Der in Pascal geschriebene Teil des Programms dient hauptsächlich dazu, die Benutzeroptionen anzuzeigen und die entsprechenden ROM-Routinen aufzurufen.

Eine Verwendung von ROM-Routinen hat zwei wesentliche Vorteile: zum einen ist die meiste Arbeit damit bereits erledigt, zum anderen sind die in Maschinencode vorliegenden Routinen blitzschnell. Eingefleischte Pascal-Freaks mögen dabei vielleicht die Nase rümpfen, Pragmatiker hingegen werden sich des im ROM plazierten Programmcodes gern bedienen.

Vor einer Beschäftigung mit der Arbeitsweise von PMU steht das Wissen um die Funktionsweise der Language Card. Unter Pascal muß die *Apple II*-Hauptplatine auf 48K RAM ausgebaut sein (Adressen $0-BFFF). Weitere 4K ($C000-CFFF) werden von Peripheriegeräten und E/A-Treibern belegt. Damit bleibt ein Adreßbereich von nur 12K ($D000-FFFF) der 16K RAM auf der Language Card frei. Dieses Problem ist zu bewältigen, indem man den Adreßbereich $D000-DFFF (4K) doppelt auslegt und zwischen beiden Banks mit Hilfe von Kontrollcodes hin- und herschaltet. Die Kontrollcodes sind spezielle Speicherzellen, die wie Schalter funktionieren und bei einem Zugriff von einer (4K) Bank auf die andere umschalten. Nähere Informationen hierzu finden Sie im *Apple Language System Manual*, Anhang D unter "Language Card Control Codes". Durch die Verwendung der Schalter, die bestimmen, auf welche der beiden 4K Banks jeweils zugegriffen werden soll, ist es möglich, 8K Bytes in einem Adreßbereich von nur 4K Bytes zu speichern. Zusätzlich zur Bank-Wechselschaltung dienen Kontrollcodes dazu,

das RAM auf der Language Card gegen ungewolltes Überschreiben zu schützen und es ein- oder auszuschalten.

Durch Abschalten des RAMs werden zwei ROMs aktiviert:

das Autostart-ROM ($F800-FFFF) auf der Language Card;
das Hauptplatinen-ROM ($D000-F7FF), das den BASIC-Interpreter enthält.

Der größte Teil des Pascal Systems liegt im RAM der Language Card ($D000-FFFF). Das BIOS (Basic Input/Output System) befindet sich in der zweiten 4K Bank ($D000-DFFF), während der P-Code-Interpreter die erste 4K Bank ($D000-DFFF) und einen Teil der restlichen 8K der Language Card belegt ($E000-FFFF). Dadurch entsteht das Problem, daß das RAM auf der Language Card nicht gelesen werden kann, wenn die ROMs aktiviert sind. Glücklicherweise greifen die meisten von PMU verwendeten Monitorroutinen nicht direkt auf das RAM der Language Card zu. Ein Teil des ROM-Disassemblers muß ins RAM kopiert werden, damit bestimmte Adressen geändert werden können (vgl. PROC INITDISASSEM in Listing Nr.2). Diese Änderungen erlauben es, das RAM jedesmal ein- oder auszuschalten, wenn wenn der ROM-Disassembler ein Byte benötigt. Die beiden 4K Banks auf der Language Card verkomplizieren die Sache noch weiter. Die Bank-Umschaltung und das Hin- und Herschalten zwischen RAM und ROM wird von den in Listing Nr.2 abgebildeten Assemblerroutinen automatisch übernommen. Trotz dieser Komplikationen hat die Vorgehensweise einen unbestreitbaren Vorteil: Durch die 12K des ROM und die zweite 4K Bank im RAM hat das Language System einen effektiv adressierbaren Speicherplatz von 80K!!

Eine weitere unschätzbare Besonderheit des PMU-Programms ist die praktisch narrensichere Dateneingabe. Unzulässige Zeichen werden nicht akzeptiert, wenn Fehler auftreten, warnt den Benutzer sofort ein unangenehmer Lautsprecherton. Selbst Längenbegrenzungen werden in der Form berücksichtigt, daß in Erwartung einer Hex-Adressen-Eingabe nur Hexadezimalziffern (0..F) zugelassen und nicht mehr als vier Zeichen akzeptiert werden.

Das PMU-Hauptmenü hält folgende Benutzerbefehle bereit:

```
PMU:D(ISASM U(NTERSUCHEN
A(ENDERN P(RINTER[AUS]
S(UCHEN M(ONITOR
B(ANK [1] E(NDE
```

D(ISASM (Disassemblieren)
Ermöglicht dem Anwender, Speicherbereiche mit dem Disassembler des Monitors zu disassemblieren.

U(NTERSUCHEN
Gibt Speicherbereiche im Hexadezimal- und ASCII-Format aus.

A(ENDERN
Ermöglicht das Ändern von Speicherinhalten.

P(RINT[EIN]
Ist der in Klammern stehende Wert "EIN", so wird die Datenausgabe auf den Drucker geleitet (80-Spalten-Format). Ist der Wert "AUS", erfolgt die Ausgabe im 40-Zeichen-Format auf dem Bildschirm. Die Wechselschaltung zwischen beiden Möglichkeiten wird über die P-Taste gesteuert.

S(UCHEN
Läßt das Auffinden einer Folge aus zwei oder drei hexadezimalen oder ASCII-Werten im Speicher für den Benutzer zum Kinderspiel werden.

M(ONITOR
Ruft den ROM-residenten Monitor auf. Das RAM der Language Card kann nicht adressiert werden, auf das gesamte übrige RAM kann man jedoch mit den Monitorbefehlen zugreifen. Der Rücksprung zu PMU erfolgt durch Eingabe von Ctrl-Y, «RETURN».

B(ANK[2] (Bankumschaltung)
Erlaubt ein Umschalten auf die andere 4K Bank. Die augenblicklich aktivierte Bank wird in eckigen Klammern angezeigt. Durch Drücken der B-Taste wird auf die jeweils andere Bank umgeschaltet.

DISASM, UNTERSUCHEN und AENDERN haben eigene Untermenüs, z.B.:

DISASM: A(UFLISTEN, W(EITER
N(EUER BEREICH B(ANK[1] E(NDE

A(UFLISTEN
: Disassembliert zwanzig Zeilen, beginnend bei einer vom Benutzer angegebenen Adresse.

W(EITER
: Disassembliert die nächsten zwanzig Zeilen.

N(EUER BEREICH
: Disassembliert einen vom Benutzer bestimmten Speicherbereich.

B(ANK
: Wie ein BANK-Befehl im PMU-Hauptmenü.

UNTERSUCHEN hat das gleiche Menü wie DISASM. Hierbei wird der angegebene Speicherbereich jedoch nicht disassembliert, sondern im Hex- und ASCII-Format ausgegeben.

Unten haben wir ein Beispiel für das AENDERN-Menü aufgeführt:

```
[ESC] = ENDE,
[CR] = ÜBERSPRINGEN)

STARTADR 400
400- A0 A0
401- B1 BE
402- C3
```

Die erste Spalte gibt die betreffende Adresse in Hex-Schreibweise an, in der zweiten Spalte steht der ursprüngliche Speicherinhalt und die dritte Spalte zeigt den neuen Wert. Durch Eingabe eines CR bleibt der aktuelle Wert unverändert, und die nächste Adresse wird ausgegeben. Durch Drücken der Esc-Taste wird die Änderungsoption verlassen und die Kontrolle wieder an die Hauptbefehlsebene von PMU zurückgegeben. Mit Ausnahme des Monitors funktioniert die Esc-Taste auch bei den anderen Befehlen von PMU. Der SUCHEN-Befehl kann dazu verwendet werden, auf einfache Weise eine Sequenz aus zwei oder drei Bytes zu finden, die hexadezimal oder als ASCII-Werte angegeben werden. Bevor SUCHEN startet, wird die zweite RAM Bank ($D000-DFFF) in die erste HIRES-Grafikseite kopiert ($2000-2FFF)

um Probleme zu vermeiden, die durch die Bank-Umschaltung entstehen könnten. Die ursprüngliche Suchroutine benötigte mindestens fünf Minuten, um die gesamten 64K RAM zu durchsuchen. Zur Verbesserung der Geschwindigkeit wurde der zeitkritische Teil der Pascal-Routine durch eine Assemblerprozedur ersetzt, mit der sich die Suchzeit auf 4,5 Sekunden verkürzen ließ (die sich jedoch entsprechend verlängert, wenn viele Übereinstimmungen gefunden werden).

SUCHEN[3]: H)EX, B(UCHST., E(NDE

Die [3] gibt an, daß nach einer Sequenz von drei Werten gesucht werden soll. Will man eine aus nur zwei Werten bestehende Hex-Sequenz finden (z.B. die Adresse $C9F7), reicht es, auf diesen Suchmodus durch Eingabe der Ziffer 2 umzuschalten. Der Bildschirm sieht dann wie folgt aus :

SUCHEN[2]: H)EX, B(UCHST., E(NDE

Drückt man jetzt die H-Taste, so entwickelt sich etwa folgender Dialog:

NACH SEQUENZ SUCHEN
1) F7 (*Bei Adressen kommt das LSB zuerst*)

2) C9

DURCHSUCHE SPEICHER...

SEQUENZ F7 C9

FF84 FFA0 FFCC (*Die gefundenen Adressen sind in Hex angegeben*)

Ein kleiner Speicherbereich (Adressen $C030-$C100) wird dabei absichtlich ausgelassen, da er *Soft Switches* enthält, mit denen verschiedene E/A-Funktionen aktiviert werden. Würden diese Adressen nicht übersprungen, resultierte dies in einer Abschaltung der normalen Textdarstellung.

Der MONITOR-Befehl von PMU eignet sich nicht dazu, das Pascal-System zu untersuchen, da der Monitor nicht auf der Language Card operieren kann (Monitor und Pascal belegen den gleichen Speicherbereich). Man kann jedoch den Monitor dazu verwenden, die Entwicklung anderer PMU-Befehle zu unterstützen, die ebenfalls ROM-Subroutinen verwenden. Man kann ihn auch dazu benutzen, Fehler in Maschinenroutinen zu suchen, die mit dem UCSD-Assembler entwickelt wurden. Mit der Sequenz: Ctrl-Y, «RETURN» gelangt man wieder ins PMU-Programm zurück. Vom Monitor aus kann man auch das BASIC aufrufen; ein Rücksprung nach Pascal ist

dann jedoch nicht mehr möglich, da BASIC die unangenehme Angewohnheit hat, den Pascal Stack durcheinander zu bringen.

Um PMU startklar zu machen, muß zuerst das in Listing Nr.1 abgedrucke Programm kompiliert und die korrekt assemblierten Routinen aus Listing Nr.2 in das Codefile eingebunden werden.

Im folgenden wollen wir Sie mit ein paar Hilfen ausstatten, mit denen es Ihnen leichter fallen sollte, das Pascal-Betriebssystem zu untersuchen. Eine gute Quelle bietet das Kapitel "Pascal Intern", in dem viele brauchbare Informationen über das Pascal-System zu finden sind.

Untenstehende Tabelle enthält einige nützliche Speicheradressen. Die in eckigen Klammern angegebenen Zahlen bezeichnen die entsprechende Speicherbank:

Tab.1 Pascal Systemadressen

Beschreibung	**Pascal 1.1-Adresse**	**Pascal II.1-Adresse**
Tastatureingabe-Puffer	3B1-3FF	3B1-3FF
(Normalerweise) Heap-Anfang	C00	C00
Zuletzt gelesenes Directory	D72-155E	D72-155E
SYSCOM	BDDE	BDDE
P-Code Decoder	D253[1]	D243[1]
P-Code-Sprungtabelle	D000[1]	D000[1]
Konsolen-Eingaberoutine (CONCK)	D772[2]	D681[2]
Konsolen-Initialisierung (CINIT)	D734[2]	D898[2]
Schreiben auf Konsole (CWRITE)	D7D0[2]	D950[2]
Schreiben auf Drucker (PWRITE)	D830[2]	D9C3[2]
Drucker-Initialisierung (PINIT)	D788[2]	D8EF[2]
Schreiben auf Disk (DWRITE)	D038[2]	D028[2]
Lesen von Disk (DREAD)	D03C[2]	D02C[2]
Disk-Initialisierung (DINIT)	D000[2]	D683[2]
Lesen von externem Gerät (RREAD)	D84E[2]	D9E5[2]
Schreiben auf externes Gerät (RWRITE)	D809[2]	D99C[2]
Initialisierung externes Gerät (RINIT)	D79C[2]	D91C[2]

Mit PMU sind kleinere Modifikationen des Pascal-Systems leicht möglich. So läßt sich zum Beispiel der Textdarstellungsmodus ändern, indem man die Adressen $DAB0[2] (Pascal 1.1) bzw. $D8ED[2] (Pascal II.1) von $40 (Normalmodus) auf blinkende ($00) oder inverse Darstellung ($40) setzt. Auch bestimmte Tastenzuordnungen lassen sich ändern :

Tab.2 Zuordnung der Pascal Ctrl-Tasten

Taste	**Hex.Wert**	**1.1 Adresse**	**II.1 Adresse**
Ctrl-K	0B	D7A2[2]	D6A5[2]
Ctrl-A	01	7A8[2]	D6AB[2]
Ctrl-Z	1A	D7BA[2]	D6BD[2]
Ctrl-F	06	BE31	BE31
Ctrl-Shift-P	00	BE32	BE32
Ctrl-S	14	BE33	BE33

Sie müssen natürlich darauf achten, daß Sie keine Tastenzuordnungen vornehmen, die mit vorher festgelegten Werten kollidieren. Da diese Änderungen im Hauptspeicher vorgenommen werden, gelten sie nur bis zur nächsten Initialisierung. Die Speicheränderung kann jedoch auch permanent gemacht werden, indem man die entsprechenden Stellen der SYSTEM.APPLE-Datei auf der Boot-Diskette ändert. Diese Methode ist relativ einfach, da die Speicheradressen $D000[2]-DFFF und $E000-FFFF hintereinander in den Blöcken 0-23 der SYSTEM.APPLE-Datei abgelegt sind, d.h. $D000[2] ist das 0-te Byte in Block 0 und $FFFF ist das letzte (511-te) Byte in Block 23. Die erste Bank ($D000[1]-DFFF) befindet sich in den Blocks 24 bis 32. Änderungen des Diskettenfiles können mit dem in diesem Buch vorgestellten

Programm PASCAL ZAP vorgenommen werden. Bei Verwendung des PASCAL ZAP-Programms müssen sie die korrekte absolute Blocknummer des zu ändernden Blocks kennen. Normalerweise ist der 0-te Block von SYSTEM.APPLE Block 6 auf der Diskette.

Da Änderungen der Disk-Dateien mehr oder weniger permanent sind, ist es besser, Systemänderungen im Hauptspeicher vorzunehmen. Gibt man dem Programm, mit dem der Speicherinhalt geändert wird, den Namen SYSTEM.STARTUP, dann wird es bei jedem Booten des Systems automatisch ausgeführt. Die Systemmodifikationen können dann einfach unwirksam gemacht werden, indem man den Namen des STARTUP-Programms ändert und die Diskette neu bootet.

Im Allgemeinen sollten Änderungen des Betriebssystems aus Gründen der Kompatibilität vermieden werden. Es gibt jedoch bestimmte Anwendungen (z.B. Darstellung von Kleinbuchstaben), wo eine Modifikation des Betriebssystems die beste (cleverste oder einfachste) Lösung des Problems darstellt. Es ist nicht allzu schwer, PMU um weitere Befehle zu erweitern. So wurde der Hex/ASCII-Suchmodus hinzugefügt, kurz nachdem das Programm veröffentlicht worden war. Ich möchte Sie als Leser dazu ermutigen, eigene Befehlsoptionen zu entwickeln oder wenigstens eigene Ideen für weitere Implementierungen beizusteuern. Je mehr Befehle hinzugefügt werden, desto nützlicher wird PMU.

Mit PMU läßt sich das Pascal System eingehend untersuchen. Es wurde nicht dazu entwickelt, um dem Benutzer Peek- und Poke-Befehle wie im BASIC zur Verfügung zu stellen, da man damit von absoluten Adressen abhängig wird und die so entstehenden Programme wenig flexibel sind. Die Absicht von PMU liegt vielmehr darin, dem Benutzer direkten Zugriff auf das Pascal System zu ermöglichen. Ich hoffe, dieses Programm hilft, die Neugierde derer zu befriedigen, die genau wissen wollen, wie das Betriebssystem funktioniert. Darüberhinaus kann das Programm als nützliches Software-Werkzeug funktionieren und denen helfen, die die Möglichkeiten des UCSD-Pascal voll ausschöpfen wollen (oder müssen).

```
(*************************************)
(* LISTING #1: PASCAL MEMORY UTILITY *)
(*                                   *)
(* VON RON DEGROAT         15-APR-81 *)
(* DEUTSCHE ADAPTION                 *)
(* BODO MESEKE                       *)
(*************************************)

(*$S+*)
PROGRAM PMU;

CONST PRINTADDR=-16128;  (* $C100 *)
      COUT1=-528;        (* $FDF0 *)

TYPE MAGIC=RECORD CASE BOOLEAN OF
            TRUE: (ADDR:   INTEGER);
            FALSE:(VECTOR:^INTEGER);
           END;

     SETOFCHAR=SET OF CHAR;

     BYTE=0..255;


VAR ENDADDR,ADDR,VAL,BANK   :INTEGER;
    NUMOFCOLS,TEMP          :INTEGER;
    IC (*INST. ZAEHLER*)    :INTEGER;
    SEARCHADDR              :INTEGER;
    EOL,BELL,CH             :CHAR;
    ESC,BS,CR               :CHAR;
    DESET,HEXSET,PMUSET     :SETOFCHAR;
    CHRSET,OKSET            :SETOFCHAR;
    GOOD,ESCOK              :BOOLEAN;
    CSW                     :MAGIC;
    HEXADDR,HEXBYTE         :STRING;
    PSW                     :STRING[3];

PROCEDURE INITDISASSEM;                    EXTERNAL;
PROCEDURE DISASSEM; (* BENUTZT IC *)       EXTERNAL;
PROCEDURE MONITOR;                         EXTERNAL;
PROCEDURE PRINTCR;                         EXTERNAL;
PROCEDURE SWITCHBANK;                      EXTERNAL;
PROCEDURE PRINTXADDR(ADDR:INTEGER);        EXTERNAL;
PROCEDURE PRINTHEXBYTE(ADDR:INTEGER);      EXTERNAL;
PROCEDURE PRINTCHARBYTE(ADDR:INTEGER);     EXTERNAL;
PROCEDURE BANK1POKE(ADDR,VAL:INTEGER);     EXTERNAL;
PROCEDURE BANK2POKE(ADDR,VAL:INTEGER);     EXTERNAL;

PROCEDURE DOCHOICE;         FORWARD;

(* ACHTUNG : Die Variable IC ist in DISASSEM *)
(* eine globale Variable, die den IC auto-   *)
(* matisch so anpasst, dass er auf die       *)
(* naechste zu disassemblierende Instruktion *)
(* zeigt.                                    *)
```

```
PROCEDURE ABORT;

BEGIN
 PAGE(OUTPUT);
 ESCOK:=FALSE;
 EXIT(DOCHOICE);
END;


FUNCTION GETCHAR(OKSET:SETOFCHAR):CHAR;

VAR CH      :CHAR;
    GOOD    :BOOLEAN;

BEGIN
 REPEAT
  READ(KEYBOARD,CH);
  IF EOLN(KEYBOARD) THEN CH:=CR;
  IF (CH=ESC) AND ESCOK THEN ABORT;
  GOOD:=CH IN OKSET;
  IF NOT GOOD THEN WRITE(BELL)
   ELSE IF CH IN [' '..CHR(125)]
          THEN WRITE(CH);
 UNTIL GOOD;
 GETCHAR:=CH;
END;


PROCEDURE GETSTRING(VAR S:STRING;
    OKSET:SETOFCHAR; MAXLEN:INTEGER);

VAR S1         :STRING[1];
    STEMP      :STRING;
    LEN        :INTEGER;
    FIRSTCHAR  :BOOLEAN;
    LASTCHAR   :BOOLEAN;
    GETSET     :SETOFCHAR;

BEGIN
 S1:=' '; STEMP:='';
 REPEAT
  LEN:=LENGTH(STEMP);
  FIRSTCHAR:=(LEN=0);
  LASTCHAR:=(LEN=MAXLEN);

  IF FIRSTCHAR THEN GETSET:=OKSET
  ELSE IF LASTCHAR THEN GETSET:=[CR,BS]
       ELSE GETSET:=OKSET+[CR,BS];

  S1[1]:=GETCHAR(GETSET);

  IF S1[1] IN OKSET THEN
   STEMP:=CONCAT(STEMP,S1)
   ELSE IF S1[1]=BS THEN
    BEGIN
     WRITE(BS,' ',BS);
     DELETE(STEMP,LEN,1);
    END;
 UNTIL S1[1]=CR;
```

```
 S:=STEMP;
END; (*GETSTRING*)

PROCEDURE PROMPTAT(L:INTEGER;S:STRING);
BEGIN
 GOTOXY(0,L);
 WRITE(S,EOL);
END;

FUNCTION DEC(HEXSTR:STRING):INTEGER;

VAR DIGIT,NUM,
    STRPTR      :INTEGER;
    XDIGITS     :PACKED ARRAY[0..15] OF CHAR;

BEGIN
 NUM:=0;
 XDIGITS:='0123456789ABCDEF';
 FOR STRPTR:=1 TO LENGTH(HEXSTR) DO
  BEGIN
   DIGIT:=SCAN(16,=HEXSTR[STRPTR],XDIGITS);
   NUM:=NUM*16+DIGIT;
  END;
 DEC:=NUM;
END;


PROCEDURE GETADDR(STR:STRING);

BEGIN
  WRITELN;WRITELN;
  WRITE(STR);
  GETSTRING(HEXADDR,HEXSET,4);
  ADDR:=DEC(HEXADDR);
END;


PROCEDURE CHANGE;

BEGIN
 PROMPTAT(0,'AENDERN: (<ESC> ');
 WRITE('= ENDE, <CR> = UEBERSPRINGEN)');
 GETADDR('STARTADR ==> ');
 PAGE(OUTPUT);
 PRINTCR;  (* AUSGABE BEGINNT LINKS *)
 REPEAT
  PROMPTAT(1,'AENDERN: (<ESC> ');
  WRITE('= ENDE, <CR> = UEBERSPRINGEN');
  GOTOXY(79,0);
  IC:=ADDR;
  PRINTXADDR(ADDR);
  PRINTHEXBYTE(ADDR);
  GOTOXY(9,23);
  GETSTRING(HEXBYTE,HEXSET+[CR],2);
  GOTOXY(79,0);
  DELETE(HEXBYTE,LENGTH(HEXBYTE),1);
  IF LENGTH(HEXBYTE)<>0 THEN
    BEGIN
      VAL:=DEC(HEXBYTE);
```

```
      IF BANK=1 THEN BANK1POKE(ADDR,VAL)
                ELSE BANK2POKE(ADDR,VAL);
    END;
  PRINTHEXBYTE(ADDR);
  PRINTCR;
  ADDR:=ADDR+1;
 UNTIL FALSE;
END; (*CHANGE*)


PROCEDURE LINEOFBYTE(VAR ADDR:INTEGER);
VAR J :INTEGER;
BEGIN
 PRINTXADDR(ADDR);
 FOR J:=ADDR TO ADDR+NUMOFCOLS-1 DO
   PRINTHEXBYTE(J);
 FOR J:=ADDR TO ADDR+NUMOFCOLS-1 DO
   PRINTCHARBYTE(J);
 ADDR:=ADDR+NUMOFCOLS; (*ADRESSE ANPASSEN*)
 PRINTCR;
END;


PROCEDURE XNEXT;
VAR I :INTEGER;
BEGIN
 PRINTCR;
 FOR I:=1 TO 20 DO LINEOFBYTE(ADDR);
END;


PROCEDURE XLIST;

BEGIN
 PROMPTAT(0,'UNTERSUCHEN : AUFLISTEN');
 GETADDR('STARTADR ==> ');
 PAGE(OUTPUT);
 XNEXT;
END;


PROCEDURE XRANGE;

BEGIN
 GETADDR('STARTADR ==> ');
 TEMP:=ADDR;
 GETADDR('ENDADR ==> ');
 PAGE(OUTPUT);
 PRINTCR;
 ENDADDR:=ADDR;
 ADDR:=TEMP;
 WHILE ADDR<=ENDADDR DO LINEOFBYTE(ADDR);
END;
```

```
PROCEDURE BANKCHANGE;

BEGIN
  BANK:=(BANK MOD 2)+1;
  SWITCHBANK;
END;


PROCEDURE EXAMINE;

VAR CH:CHAR;

BEGIN
 REPEAT
  PROMPTAT(0,'UNTERSUCHEN: A(UFLISTEN W(EITER ');
  WRITE('N(EUER BEREICH B(ANK[',BANK,'] E(NDE');
  CH:=GETCHAR(DESET);
  PAGE(OUTPUT);
  CASE CH OF
   'A','a':XLIST;
   'W','w':XNEXT;
   'N','n':XRANGE;
   'B','b':BANKCHANGE;
  END;
 UNTIL CH IN ['E','e'];
END; (*EXAMINE*)


PROCEDURE NEXT;
VAR I :INTEGER;
BEGIN
  FOR I:=1 TO 20 DO DISASSEM(* IC *);
  PRINTCR;
END;


PROCEDURE LIST;

BEGIN
 PROMPTAT(0,'DISASM: AUFLISTEN');
 GETADDR('STARTADR ==> ');
 PAGE(OUTPUT);
 IC:=ADDR;     (*IC (INSTR COUNTER)*)
 NEXT;         (*GLOBAL FUER DISASSEM*)
END;


PROCEDURE RANGE;

BEGIN
  PAGE(OUTPUT);
  PROMPTAT(0,'DISASM: BEREICH');
  GETADDR('STARTADR ==> ');
  IC:=ADDR;
  GETADDR('ENDADR ==> ');
  PAGE(OUTPUT);
  WHILE IC<=ADDR DO DISASSEM(* IC *);
  PRINTCR;
END;
```

```
PROCEDURE DISASM;

VAR CH:CHAR;

BEGIN
 REPEAT
  PROMPTAT(0,'DISASM: A(UFLISTEN W(EITER ');
  WRITE('N(EUER BEREICH B(ANK[',BANK,'] E(NDE');
  CH:=GETCHAR(DESET);
  PAGE(OUTPUT);
  CASE CH OF
   'A','a':LIST;
   'W','w':NEXT;
   'N','n':RANGE;
   'B','b':BANKCHANGE;
  END; (*CASE*)
 UNTIL CH IN ['E','e'];
END;


PROCEDURE PRINT;

BEGIN

(*$I-*)
  UNITCLEAR(6);
  IF IORESULT<>0 THEN
   BEGIN
    PROMPTAT(3,'DRUCKER NICHT ANGESCHLOSSEN');
    EXIT(PRINT);
   END;
(*$I+*)

  IF PSW='AUS' THEN
   BEGIN
    CSW.VECTOR^:=PRINTADDR;
    PSW:='EIN';
    NUMOFCOLS:=16;
   END
  ELSE
   BEGIN
    CSW.VECTOR^:=COUT1;
    PSW:='AUS';
    NUMOFCOLS:=8;
   END;
END; (*PRINT*)


FUNCTION FOUND(FIRST,SECOND,THIRD:BYTE;
               NOTHIRD:BOOLEAN):BOOLEAN;
EXTERNAL;
PROCEDURE MOVEBANK2;
EXTERNAL;
```

```
(******************************************)
(* DURCHSUCHEN DES HAUPTSPEICHERS NACH EINER*)
(* FOLGE VON 2 ODER 2 HEXADEZIMALEN ODER    *)
(* ASCII-ZEICHEN                            *)
(******************************************)

PROCEDURE SEARCHMEM;

VAR FIRST,SECOND,THIRD     :BYTE;
    SFIRST,SSECOND,STHIRD :STRING;
    SLEN,
    FOR2OR3,COLCOUNT       :INTEGER;
    NOTHIRD,
    SEQNOTFOUND,
    DONE                  :BOOLEAN;
    CH                    :CHAR;
    XADDR                 :STRING[4];


BEGIN
 NOTHIRD:=FALSE;
 THIRD:=0;
 STHIRD:='0';
 FOR2OR3:=3;

 REPEAT
 GOTOXY(0,0);
 WRITE('>SUCHEN[',FOR2OR3,']: H(EX, B(UCHST., E(NDE');
 CH:=GETCHAR(['H','h','B','b','E','e','2','3']);
 PAGE(OUTPUT);

 IF CH IN ['2','3'] THEN
  CASE CH OF
   '2':BEGIN
        NOTHIRD:=TRUE;
        FOR2OR3:=2;
       END;
   '3':BEGIN
        NOTHIRD:=FALSE;
        FOR2OR3:=3;
       END;
  END (*CASE*)

  ELSE
   IF NOT (CH IN ['E','e']) THEN
    BEGIN

     CASE CH OF
      'H','h':BEGIN OKSET:=HEXSET; SLEN:=2; END;
      'B','b':BEGIN OKSET:=CHRSET; SLEN:=1; END;
     END; (*CASE*)

(*SEQUENZ HOLEN*)

     WRITELN;
     WRITELN('NACH SEQUENZ SUCHEN');
     WRITE('1. => ');
     GETSTRING(SFIRST,OKSET,SLEN);WRITELN;
     WRITE('2. => ');
     GETSTRING(SSECOND,OKSET,SLEN);WRITELN;
```

```
     IF NOTHIRD=FALSE THEN
      BEGIN
       WRITE('3. ==> ');
       GETSTRING(STHIRD,OKSET,SLEN);WRITELN;
      END;

(*IN INTEGER UMWANDELN*)

     IF CH IN ['H','h'] THEN
      BEGIN
       FIRST :=DEC(SFIRST);
       SECOND:=DEC(SSECOND);
       THIRD :=DEC(STHIRD);
      END

     ELSE
      BEGIN
       FIRST :=ORD(SFIRST[1]);
       SECOND:=ORD(SSECOND[1]);
       THIRD :=ORD(STHIRD[1]);
      END;

     PROMPTAT(12,'DURCHSUCHE SPEICHER...');WRITELN;
     WRITELN('ZWEITE BANK  (D000-DFFF) IN GRAPHIKSEITE');
     WRITE('KOPIERT (2000-2FFF)');

(* DIE EIGENTLICHE SUCHE ERFOLGT IN DER *)
(* EXTERNEN PROZEDUR FOUND              *)

     MOVEBANK2;
     SEQNOTFOUND:=TRUE;
     SEARCHADDR:=0;           (* SUCHE BEI NULL BEGINNEN *)
     COLCOUNT  :=0;
     DONE:=FALSE;


     GOTOXY(0,20);
     WRITE('SEQUENZ',SFIRST:4,SSECOND:4);
     IF NOT NOTHIRD THEN
      WRITELN(STHIRD:4);

     GOTOXY(80,0); (*CURSOR VOM BILDSCHIRM NEHMEN*)
     PRINTCR;      (*AUSGABE LINKS BEGINNEN*)

     REPEAT
      IF (FOUND(FIRST,SECOND,THIRD,NOTHIRD)) THEN
       BEGIN
        SEQNOTFOUND:=FALSE;
        PRINTXADDR(SEARCHADDR);
        SEARCHADDR:=SEARCHADDR+1;
        COLCOUNT:=(COLCOUNT+1) MOD NUMOFCOLS;
        IF (COLCOUNT=0) THEN
          IF (PSW='EIN') THEN PRINTCR;
       END
      ELSE
        DONE:=TRUE;
     UNTIL DONE;

     IF SEQNOTFOUND THEN PROMPTAT(22,'NICHT GEFUNDEN')
                    ELSE PRINTCR;
```

```
      WRITE(BELL);
    END;
 UNTIL CH IN ['E','e'];
END; (*SEARCH*)


PROCEDURE DOCHOICE;

BEGIN
 PAGE(OUTPUT); ESCOK:=TRUE;
 CASE CH OF
  'D','d':DISASM;
  'U','u':EXAMINE;
  'A','a':CHANGE;
  'S','s':SEARCHMEM;
  'B','b':BANKCHANGE;
  'P','p':PRINT;
  'M','m':MONITOR;
 END;
 ESCOK:=FALSE;
END;


PROCEDURE INITIALIZE;

BEGIN
 BELL:=CHR(7);          (*CTL-G*)
 EOL:=CHR(29);          (*CTL-]*)
 ESC:=CHR(27);          (*CTL-[*)
 BS:=CHR(8);            (*CTL-H*)
 CR:=CHR(13);           (*CTL-M*)

 ESCOK:=FALSE;
 BANK:=1;
 PSW:='AUS';   (*PRINTER SCHALTER*)
 NUMOFCOLS:=8;
 ADDR:=0;
 CSW.ADDR:=54; (*ZEICHENAUSGABESCHALTER*)

 DESET:=['A','a','N','n','W','w',
         'B','b','E','e'];

 PMUSET:=['D','d','U','u','B','b','M','m',
          'S','s','A','a','P','p','E','e'];

 CHRSET:=[' '..CHR(127)];

 HEXSET:=['0'..'9']+['A'..'F'];

 INITDISASSEM;

END; (*INIT*)

BEGIN (*HAUPTPROGRAMM*)

 INITIALIZE; (*ALLES*)
```

```
 REPEAT
  PROMPTAT(0,'PMU: D(ISASM U(NTERS ');
  WRITE('A(ENDERN P(RINTER[',PSW,'] M(ONITOR ');
  WRITE('S(UCHEN B(ANK[',BANK,'] E(NDE');
  CH:=GETCHAR(PMUSET);
  DOCHOICE;
 UNTIL CH IN ['E','e'];

END.
```

```
;LISTING #2: PMU.PROC-----------------
;
;VON RON DEGROAT       4/81
;-------------------------------------

;MONITOR ROM ADRESSEN

SETNORM .EQU 0FE84  ;E/A INITIALISIERUNG
INIT    .EQU 0FB2F
SETVID  .EQU 0FE93
SETKBD  .EQU 0FE89

PRBYTE  .EQU 0FDDA  ;ZEICHENAUSGABE
COUT    .EQU 0FDED  ;UNTERROUTINEN
PRNTYX  .EQU 0F940

PCADJ   .EQU 0F953
PC      .EQU 0003A  ;PROGRAMMZAEHLER
MON     .EQU 0FF65  ;MONITOR

RETURN  .EQU 00000  ;PASCAL RET ADDR
ADDR    .EQU 00002  ;UEBERGEBENER PARAMETER
VAL     .EQU 00004  ;UEBERGEBENER PARAMETER

;DATEN ODER ROUTINE LADEN (MAXIMAL 256 BYTES)

        .MACRO LOAD  ;FORMAT:
        LDY #00      ;LADE SOURCE,DEST,LAEN
$1      LDA %1,Y
        STA %2,Y
        INY
        CPY #%3
        BNE $1
        .ENDM

;ADRESSE VOM STACK HOLEN

        .MACRO POP  ;FORMAT: POP ADDR
        PLA
        STA %1
        PLA
        STA %1+1
        .ENDM

;ADRESSE AUF STACK ABLEGEN
```

```
        .MACRO PUSH ;FORMAT: PUSH ADDR
        LDA %1+1
        PHA
        LDA  %1
        PHA
        .ENDM

        .MACRO ROMSELECT
        STA 0C08A
        .ENDM

        .MACRO RAMSELECT
        JSR BANK
        .ENDM


        .PROC BANK1POKE,2

        .DEF POKE

        STA 0C089       ;ERSTE 4K BANK
                        ;BESCHREIBBAR MACHEN
POKE    POP RETURN
        POP VAL
        POP ADDR
        LDA VAL
        LDY #00
        STA (ADDR),Y    ;POKE VAL IN ADDR
        STA 0C088       ;RAM SCHREIBSCHUETZEN
                        ;2. BANK ANWAEHLEN
        PUSH RETURN
        RTS


        .PROC BANK2POKE,2

        .REF POKE

        STA 0C081       ;ZWEITE 4K BANK BESCHREIBBAR MACHEN
        JMP POKE

        .PROC MONITOR

        LOAD USER,03F8,03 ;CTL-Y JUMP
        ROMSELECT
        JMP MON
RET     PLA          ;CTL-Y ADDR LOESCHEN
        PLA
        STA 0C088    ;RAM BANK1 EINSCHALTEN
        RTS
USER    JMP RET


        .PROC PRINTCR
        ROMSELECT
        LDA #8D      ;<RET>
        JSR COUT
        STA 0C088
        RTS
```

```
        .PROC PRINTXADDR,1

        POP RETURN
        PLA          ;HOLE ADRESSE
        TAX
        PLA
        TAY
        ROMSELECT
        JSR PRNTYX ;HEX ADDR AUSGEBEN
        LDA #0A0   ;EIN LEERZEICHEN AUSGEBEN
        JSR COUT
        STA 0C088
        PUSH RETURN
        RTS


        .PROC PRINTHEXBYTE,1

        .REF BANK  ;WIRD VON RAMSELECT BENUTZT

        POP RETURN
        POP ADDR
        RAMSELECT
        LDY#00
        LDA (ADDR),Y ;HEX BYTE HOLEN
        ROMSELECT
        JSR PRBYTE ;HEX BYTE AUSGEBEN
        LDA #0A0   ;LEERZEICHEN AUSGEBEN
        JSR COUT
        STA 0C088
        PUSH RETURN
        RTS


        .PROC PRINTCHARBYTE,1

        .REF BANK

        POP RETURN
        POP ADDR
        RAMSELECT
        LDY #00
        LDA (ADDR),Y;ZEICHENBYTE HOLEN
        ORA #80      ;HI BIT SETZEN
        CMP #0A0     ;CONTROLZEICHEN AUSGEBEN
        BCS NORMAL   ;ZEICHEN ALS '.'
        LDA #0AE     ; '.'
NORMAL  ROMSELECT
        JSR COUT
        STA 0C088
        PUSH RETURN
        RTS


        .PROC SWITCHBANK

        .DEF BANK
```

```
         LDA BANK+1   ;0C088 IN
         EOR #08      ;0C080 AENDERN UND
         STA BANK+1   ;UMGEKEHRT
         RTS
BANK     STA 0C088
         RTS

         .PROC DISASSEM

         .PUBLIC IC         ;BEFEHLSZAEHLER
         .REF INSTDSP,BANK

         LDA IC        ;IC
         STA PC        ;NACH PC UEBERTRAGEN
         LDA IC+1
         STA PC+1
         ROMSELECT
         JSR INSTDSP   ;EINEN BEFEHL DISASSEMBLIEREN
         JSR PCADJ     ;PC ANPASSEN
         STA PC
         STY PC+1
         STA IC        ;IC ANPASSEN
         STY IC+1
         STA 0C088     ;VOR RUECKSPRUNG AUF BANK1
         RTS           ;ZURUECKSCHALTEN


         .PROC INITDISASSEM

         .DEF INSTDSP

         .REF BANK

         JMP BEGIN     ;KOPIERBEREICH-UEBERSPRINGEN

;HIER HAUPTTEIL DES ROM DISASSEMBLERS KOPIEREN,
;DAMIT ER FUER DIE FUNKTION UNTER PASCAL
;MODIFIZIERT WERDEN KANN

INSDS1   .BLOCK ODF     ;KOPIE BEGINNT HIER

BEGIN    ROMSELECT
         LOAD OF882,INSDS1,ODF ;KOPIEREN

INSTDSP .EQU INSDS1+04E
PATCH1   .EQU INSDS1+00A
PATCH2   .EQU INSTDSP
PATCH3   .EQU INSDS1+0B0

;NOTWENDIGE AENDERUNGEN VORNEHMEN UND VIDEO INITIALISIEREN

         LOAD CHANGE1,PATCH1,03
         LOAD CHANGE2,PATCH2,09
         LOAD CHANGE3,PATCH3,04
         JSR SETNORM
         JSR INIT
         JSR SETVID
         JSR SETKBD
         RAMSELECT  ;LANGUAGE CARD EINSCHALTEN
         RTS
```

```
CHANGE1 JSR DIS1

CHANGE2 JSR INSDS1
        PHA
        JSR DIS2
        NOP
        NOP

CHANGE3 JSR DIS3
        NOP

DIS1    RAMSELECT  ;LANGUAGE CARD EINSCHALTEN
        LDA (PC,X) ;OPCODE BYTE HOLEN
        ROMSELECT  ;ROM EINSCHALTEN
        TAY
        RTS

DIS2    RAMSELECT
        LDA(PC),Y
        ROMSELECT
        JMP PRBYTE

DIS3    CMP #0E8
        RAMSELECT
        LDA (PC),Y
        ROMSELECT
        RTS


;-------------------------------------
;
; DIE FOLGENDEN EXTERNEN PROZEDUREN UND
; FUNKTIONEN WERDEN VON SEARCHMEMORY
; VERWENDET
;
;-------------------------------------


;-------------------------------------
;FUNCTION FOUND(ADDR:INTEGER; FIRST,
;               SECOND,THIRD:BYTE;
;               NOTHIRD:BOOLEAN):BOOLEAN;
;(BYTE:0..255)
;
;VON RON DEGROAT    15-JUN-80
;-------------------------------------


         .FUNC FOUND,4

         .MACRO PULL ;PARAMETER HOLEN UND RETTEN
         PLA
         STA %1
         PLA
         .ENDM

         .MACRO FIND
         LDA @SRCHADDR,Y
         CMP %1
```

```
        BNE NOMATCH
        INY          ;Y AUF NAECHSTES BYTE SETZEN
        .ENDM

THIRD    .EQU 12
SECOND   .EQU 13     ;ZU SUCHENDE SEQUENZ
FIRST    .EQU. 15
NOTHIRD  .EQU 14     ;DRITTES BYTE VORHANDEN ?
SRCHADDR .EQU 16     ;SUCHADRESSE
ORIGIN   .EQU 18     ;STARTADRESSE FUER SUCHE

        .PUBLIC SEARCHADDR

        POP RETURN ;PASCAL RUECKSPRUNGADRESSE

        PLA          ;OFFSET UEBERSPRINGEN
        PLA
        PLA
        PLA

        PULL NOTHIRD ;PARAMETER HOLEN
        PULL THIRD   ;UND WERTE SPEICHERN
        PULL SECOND
        PULL FIRST

        LDA #00      ;MSB DES FUNKTIONSERGEBNISSES AUF STACK
        PHA
        STA ORIGIN   ;START BEI NULL
        STA ORIGIN+1

        LDA SEARCHADDR   ;PUBLIC ADDR
        STA SRCHADDR     ;ZERO PAGE ADRESSE
        LDA SEARCHADDR+1
        STA SRCHADDR+1

LOOP    LDY #00      ;Y JEDESMAL INITIALISIEREN
        FIND FIRST   ;PRUEFEN OB UEBEREINSTIMMUNG
        FIND SECOND  ;NUR WENN ERSTES BYTE STIMMT
        LDA NOTHIRD  ;DRITTES BYTE ZU VERGLEICHEN ?
        CMP #01
        BEQ MATCH    ;WENN KEIN DRITTES BYTE DANN UEBEREINSTIMMUNG
        FIND THIRD   ;SONST DRITTES BYTE PRUEFEN

MATCH   LDA #01      ;FOUND = TRUE SETZEN
        CLC          ;GOTO END
        BCC END

NOMATCH INC SRCHADDR ;LSB DER SUCHADRESSE ERHOEHEN
        BNE NOBUMP   ;SPRUNG WENN KEIN UEBERTRAG

        INC SRCHADDR+1 ;MSB DER SUCHADRESSE ERHOEHEN

NOBUMP  LDA SRCHADDR+1 ;WENN MSB DER SUCHADRESSE =
        CMP ORIGIN+1   ;MSB DES URSPRUNGS,
        BNE NOTDONE
        LDA SRCHADDR   ;DANN PRUEFE LSB
        CMP ORIGIN
        BEQ ABORT      ;WENN = DANN ABBRUCH
```

```
NOTDONE  LDA SRCHADDR+1 ;SOFT SWITCHES UEBERSPRINGEN
         CMP #0C0       ;$C030 BIS $C100
         BNE LOOP
         LDA SRCHADDR
         CMP #30
         BCC LOOP
         LDA #0C1
         STA SRCHADDR+1
         LDA #00
         STA SRCHADDR

         CLC
         BCC LOOP       ;NOCHMAL

ABORT    LDA #00        ;RESULTAT AUF FALSE SETZEN
END      PHA            ;LSB DES RESULTATS AUF STACK
         LDA SRCHADDR
         STA SEARCHADDR
         LDA SRCHADDR+1
         STA SEARCHADDR+1
         PUSH RETURN
         RTS            ;ZURUECK INS PASCALPROGRAMM


         .PROC MOVEBANK2

         LDA #0D0    ;(0000) = D000
         STA 01
         LDA #20     ;(0002) = 2000
         STA 03
         LDA #00
         STA 00
         STA 02
         TAY
         TAX
         LDA 0C083   ;ZWEITE BANK ANSPRECHEN
LOOP     LDA (00),Y ;D000-DFFF
         STA (02),Y ;NACH 2000-2FFF VERSCHIEBEN
         INY
         BNE LOOP
         INC 01      ;SEITENNUMMER DES STARTS
         INC 03      ;SEITENNUMMER DES ZIELES
         INX
         CPX #10     ;16 SEITEN UEBERTRAGEN
         BNE LOOP
         LDA 0C088   ;RAM SCHREIBSCHUETZEN
         RTS

         .END
```

William Janes

RECOVER

```
PROGRAM RECOVER;

(* VON W. JANES, 28.12.1980                                               *)

(* DIESES PROGRAMM VERWENDET DIE MASCHINENNAHE PROZEDUR "UNITREAD"  *)
(* ZUM ZUM LESEN VON DATEIEN, DEREN DIRECTORY DEFEKT IST UND WURDE  *)
(* HAUPTSAECHLICH DAZU ENTWICKELT, TEXTDATEIEN WIEDERHERZUSTELLEN.  *)
(* DIE SO REKONSTRUIERTEN FILES KOENNEN AUF DEN BILDSCHIRM, DEN     *)
(* DRUCKER ODER IN EIN DISKETTENFILE GESCHRIEBEN WERDEN. AUF DIE    *)
(* DATEIEN WIRD MIT HILFE EINER ANFANGSBLOCKNUMMER UND DER ZU LESEN-*)
(* DEN BLOCKANZAHL ZUGEGRIFFEN. NICHT ANZEIGBARE STEUERZEICHEN      *)
(* KOENNEN ENTWEDER UNTERDRUECKT ODER MIT IHREN IN KLAMMERN         *)
(* GESETZTEN ASCII-WERTEN AUSGEGEBEN WERDEN.                        *)


TYPE OUTMODE = (PRNTR, CONSL, DISC);
     INMODE  = 4..5;
     BLOCK   = PACKED ARRAY [0..511] OF CHAR;
     CHARSET = SET OF CHAR;

VAR START,STOP: INTEGER;
    AGAIN     : CHAR;
    FILTRATION: BOOLEAN;
    INDENTBYTE: BOOLEAN;
    OUTDEVICE : OUTMODE;
    INDEVICE  : INMODE;
    OUTFILE   : TEXT;
    PRINTSET  : SET OF CHAR;

FUNCTION PROMPT(S:STRING; OKSET:CHARSET):CHAR;

VAR GOOD: BOOLEAN;
      CH: CHAR;

BEGIN
  REPEAT
    WRITELN;
    WRITE(S);
    READ(CH);
    WRITELN;
```

```
    UNITCLEAR(1);
    GOOD:=CH IN OKSET;
    IF NOT GOOD THEN
      WRITE(CHR(7));
  UNTIL GOOD;
  PROMPT:=CH;
END;

PROCEDURE WRITEBLOCK(BLKNUM:INTEGER; BLOCKARRAY:BLOCK);

VAR I      :INTEGER;
    TEMP   :CHAR;
    NOSKIP :BOOLEAN;

BEGIN
  IF OUTDEVICE <> DISC
    THEN BEGIN
         WRITELN(OUTFILE,CHR(13));
         WRITELN(OUTFILE,'BLOCKNUMMER ',BLKNUM);
         WRITELN(OUTFILE);
       END;

  FOR I:=0 TO 511 DO
    BEGIN
      TEMP:=BLOCKARRAY[I];
      IF (TEMP IN PRINTSET) AND NOT INDENTBYTE
        THEN WRITE(OUTFILE,TEMP)
        ELSE BEGIN
          IF FILTRATION
            THEN INDENTBYTE:=(TEMP=CHR(16))
            ELSE WRITE(OUTFILE,'[',ORD(TEMP),']')
        END
    END; (*I SCHLEIFE*)

  IF OUTDEVICE <> DISC
      THEN WRITELN(OUTFILE,CHR(13),'ENDE VON BLOCK NR. ',BLKNUM);
END; (*WRITEBLOCK*)

PROCEDURE GETPRNTOPTION(VAR RSLT:BOOLEAN);

VAR CH: CHAR;

BEGIN
  WRITELN;
  WRITELN('AUSGABEOPTIONEN:');
  WRITELN('           A)LLE ZEICHEN AUSGEBEN (FALLS NOETIG,ORD-WERTE)');
  WRITELN('           H)ERAUSFILTERN DER NICHT DRUCKBAREN ZEICHEN');
  CH:=PROMPT('IHRE WAHL ? ',['A','H']);
  RSLT:=(CH='H');
END; (*GETPRINTOPTION*)

PROCEDURE INITIALIZE(VAR PRNTSET:CHARSET; VAR INDEV:INMODE;
                     VAR OUTDEV:OUTMODE; VAR FILTER:BOOLEAN);

VAR SELECTION, DEVICE: CHAR;

BEGIN
  DEVICE:=PROMPT('ZU LESENDE UNIT (4..5)? ',['4','5']);
  IF DEVICE='4'
```

```
      THEN INDEV:=4
      ELSE INDEV:=5;
   SELECTION:=PROMPT('AUSGABE AUF P)RINTER B(ILDSCHIRM D)ISKETTE ',
                     ['P','B','D']);
   CASE SELECTION OF
      'P':BEGIN
             PRNTSET:=[CHR(32)..CHR(127),CHR(13)];
             REWRITE(OUTFILE,'PRINTER:');
             OUTDEV:=PRNTR;
             GETPRNTOPTION(FILTER);
          END;
      'B':BEGIN
             PRNTSET:=[CHR(32)..CHR(127),CHR(13)];
             REWRITE(OUTFILE,'CONSOLE:');
             OUTDEV:=CONSL;
             GETPRNTOPTION(FILTER);
          END;
      'D':BEGIN
             PRNTSET:=[CHR(32)..CHR(127),CHR(13),CHR(16)];
             OUTDEV:=DISC;
             FILTER:=TRUE
          END
      END (*CASE*)
END; (*INITIALIZE*)


PROCEDURE PROCESSBLOCKS(FIRST,LAST:INTEGER);

VAR BLOCKNUM:INTEGER;
    BUFFER  :BLOCK;
    F       :STRING;

BEGIN
  IF OUTDEVICE = DISC
    THEN BEGIN
           WRITELN;
           WRITE('SPEICHERN ALS? ');
           READLN(F);
           F:=CONCAT(F,'.TEXT');
           REWRITE(OUTFILE,F);
         END;

   INDENTBYTE:=FALSE;
   (* WIRD ZUM LOESCHEN VON [INDENT] NACH DEM [DLE] ZEICHEN BENUTZT *)

   FOR BLOCKNUM:=FIRST TO LAST DO
     BEGIN
       UNITREAD(INDEVICE,BUFFER,512,BLOCKNUM);
       WRITEBLOCK(BLOCKNUM,BUFFER);
     END; (*SCHLEIFE*)
   IF OUTDEVICE=DISC THEN CLOSE(OUTFILE,LOCK);
 END; (*PROCESSBLOCKS*)

 PROCEDURE GETRANGE(VAR FIRST,LAST:INTEGER);

 VAR MAX,NUM:INTEGER;
```

```
BEGIN
  WRITELN;
  WRITE('START BEI BLOCK? ');
  READLN(FIRST);
  MAX:=280-FIRST;
  WRITELN;
  WRITE('BLOCKANZAHL? (1..',MAX,') .');
  READLN(NUM);
  LAST:=FIRST+NUM-1;
END; (*GETRANGE*)

BEGIN (*HAUPTPROGRAMM*)
   INITIALIZE(PRINTSET,INDEVICE,OUTDEVICE,FILTRATION);
   REPEAT
     GETRANGE(START,STOP);
     PROCESSBLOCKS(START,STOP);
     AGAIN:=PROMPT('NOCH EINEN DISKBEREICH BEARBEITEN? ',['J','N']);
   UNTIL AGAIN = 'N';
END.
```

Chris Wilson

Pascal 1.1 Speicher-nutzung

Die Informationen in der folgenden Tabelle sind in mehrere Abschnitte unterteilt (z.B. Zero-Page-Variablen). Die Einrückung soll das Auffinden der einzelnen Informationen erleichtern. Indirekte Adressierung wird im folgenden durch das @-Zeichen repräsentiert!

Adresse(n)	Bedeutung
0000-00FF	Zero-Page mit Interpreter- und BIOS-Variablen
0050/1	BASE Basisprozedur
0052/3	MP Stackzeiger
0054/5	JTAB Sprungtabellenzeiger
0056/7	SEG Segmentzeiger
0058/9	IPC Interpreter-Programmzähler
005A/B	NP Neuer Zeiger
005C/D	KP Programm-Stack-Zeiger
006E/0070	Indirekter Sprung, wird vom Interpreter benutzt (JMP @D0XX)
0071-0073	Indirekter Sprung, der vom CSP-Befehl benutzt wird (JMP @D1XX)
007E	NOSPEC Kontrollwort von UNITREAD/WRITE
0080	NOCRLF Kontrollwort von UNITREAD/WRITE
0096-00A2	DLE Expansionstabelle, wird von UNITRAD/WRITE verwendet
00BD/E	ZEROL u. ZEROH, zum Löschen des Speichers bei RESET
00BF/C0	JUMP1 u.JUMP2, Einsprungadressen beim Auftreten von Kontrollzeichen
00C1/2	BXS1L und BXS1H
00C3/4	BXS2L und BXS2H
00C5/6	CKPTRL u. CKPTRH
00D0/1	CHECKL u.CHECKH

00D2	TT1
00D3	NT2
00D4	TT3
00E0	HSMODE wird von HI-RES-Routinen verwendet
C0E1	HCMODE wird von HI-RES-Routinen verwendet
00E2/3	ACJVAFLD Zeiger auf ATTACH-Kopie des Original-BIOS-Sprungvektors
00E4/5	RTPTR Zeiger auf Zeichen der Gerätelesetabelle
00E6/7	WTPTR Zeiger auf Zeichen der Geräteschreibtabelle
00E8/9	UDJVP Zeiger auf Sprungvektor für User Device
00EA/B	DISKNUMP Zeiger auf Diskettennummer-Vektor
00EC/D	JVBFOLD Zeiger auf BIOS-Sprungvektor vor Fold-Operation
00EE/F	JVAFOLD Zeiger auf BIOS-Sprungvektor nach Fold-Operation
00F0/1	BAS1L u. BAS1H Zeiger auf Textseite 1
00F2/3	BAS2L u. BAS2H Zeiger auf Textseite 2
00F4	CH Horizontalposition des Cursors
00F5	CV Vertikalposition des Cursors
00F6	TEMP1
00F7	TEMP2
00F8/9	SYSCOM (Zeiger auf SYSCOM)
0100-01FF	Berechnungsstack für Interpreter
0200-????	BUFFER BIOS Diskettenpuffer
03B1-03FF	BIOS Tastaturpuffer
0400-07FF	APPLE Textseite 1
0800-0BFF	APPLE Textseite 2
0C00-????	Heap
0C40-0C7D	INPUT-Datei-Informationsblock
0C7E-0CBB	OUTPUT-Datei-Informationsblock
0CBC-0CF9	KEYBOARD-Datei-Informationsblock
0CFA-0D35	Datei-Informationsblock für SYSTEM.WRK.TEXT (falls vorhanden)
0D36-0D71	Datei-Informationsblock für SYSTEM.WRK.CODE (falls vorhanden)
A988-AC99	Activation Record für Segment 0, Prozedur 1 von SYSTEM.PASCAL
A994/5	Zeiger auf SYSCOM
A996/7	Zeiger auf Informationsblock der INPUT-Datei
A998/9	Zeiger auf Informationsblock der OUTPUT-Datei
A99A/B	Zeiger auf Datei-Informationsblock von KEYBOARD
A9A2/3	Zeiger auf Informationsblock von SYSTEM.WRK.CODE
A9A4/5	Zeiger auf Informationsblock von SYSTEM.WRK.TEXT

A9AC/D	"Student"-Flag aus MISCINFO
A9AE/F	"Slow Terminal"-Flag aus MISCINFO
A9B0/1	Editor-Escape-Taste aus MISCINFO
A9B2/3	Flag für das Vorhandensein von SYSTEM.WRK.CODE
A9B4/5	Flag für das Vorhandensein von SYSTEM.WRK.TEXT
A9B6-A9BD	Laufwerkname von SYSTEM.WRK.CODE (STRING[7])
A9BE-A9C5	Laufwerkname von SYSTEM.WRK.TEXT (STRING[7])
A9C6-A9CD	Laufwerkname von permanentem Workfile (String[7])
A9CE-A9DD	Dateiname der SYSTEM.WRK.CODE-Datei (STRING[15])
A9DE-A9ED	Dateiname der SYSTEM.WRK.TEXT-Datei (STRING[15])
A9EE-A9FD	Dateiname der permanenten Work-Datei (STRING[15])
A9FE/F	Zeiger auf leeren Heap für Benutzerprogramme
AA02/3	Zeiger auf KEYBOARD-Datei-Informationsblock
AA04/5	Zeiger auf OUTPUT-Datei-Informationsblock
AA06/7	Zeiger auf INPUT-Datei-Informationsblock
AA08-AA0F	Standard (:) Laufwerkname (STRING[7])
AA10-AA17	System (*) Laufwerkname (STRING[7])
AA18/9	Tagesdatum in folgender Form:

```
PACKED RECORD
MONAT: 0..12;
TAG : 0..31;
JAHR : 0..99;
 END;
```

AA1E-AA6F	Befehlszeile des Command-Levels (STRING[80])
AA70-AA79	Tabelle für Integer-Zehnerpotenzen (ARRAY [0..4] OF INTEGER)
AA7A-AA85	String mit Nullzeichen für die Verzögerung von Vertikalbewegungen - aus MISCINFO (STRING [11])
AA86-AA8D	Ziffern (SET OF "0".."9")
AA8E-AB29	Gerätetabelle

```
ARRAY [0..12] OF RECORD
GNAME: STRING [7]
 (* Gerätename *)
CASE BLOCKIERT: BOOLEAN OF
 (* True, wenn Gerät blockiert *)
TRUE: (LETZTBLOCK: INTEGER)
 (* Letzter Block, wenn Gerät blockiert *)
 END;
```

AB2A-AB41	Dateiname des SYSTEM.ASSEMBLER (STRING[23])

AB42-AB59	Dateiname des SYSTEM.COMPILER (STRING[23])
AB5A-AB71	Dateiname des SYSTEM.EDITOR (STRING[23])
AB72-AB89	Dateiname des SYSTEM.FILER (STRING[23])
AB8A-ABA1	Dateiname des SYSTEM.LINKER (STRING[23])
ABA6-ABC5	Konfigurationszeichen von MISCINFO (SET OF CHAR)
ABC6-ABDD	Name der nächsten Datei von SETCHAIN aus CHAINSTUFF (STRING[23])
ABDE-AC2F	Nachricht von SETCVAL aus CHAINSTUFF (STRING[80])
AC3A/B	Flag, das gesetzt wird, wenn eine EXEC-Datei gelesen wird
AC3C/D	Flag, das beim Schreiben von EXEC-Dateien gesetzt ist
AC3E/F	System-Swapping-Flag
AC44/5	Flag, das unmittelbar nach dem Booten gesetzt ist
AC5A-AC61	Laufwerkname von EXEC-Datei (STRING[7])
AC70-Ac7F	Dateiname von EXEC-File (STRING[15])
AC9A-BD1B	Code von Segment 0, Prozeduren 29 bis 57 von SYSTEM.PASCAL
BD1E-BD5D	Tabelle für Interpreter-Segmentverwendungsanzahl
BD5E-BD9D	Interpreter-Segmentadressentabelle
BD9E-BDDD	Segmentprozedurverzeichnis für Interpreter
BDDE-BEBB	SYSCOM
DEEE/F	Code für IORESULT
BDE0/1	Fehlercode für Exec Errors (Laufzeitfehler)
BDE2/3	Boot Unit (Kaltstart-Laufwerk)
BDE4/5	Debugger Status
BDE6/7	Zeiger auf zuletzt gelesenes Diskettenverzeichnis
BDE8/9	Zeiger auf EXEC ERROR Activation Record
BDEA/B	Kopie des Interpreter-BASE-Registers
BDEC/D	Kopie des Interpreter-MP-Registers
BDEE/F	Kopie des Interpreter-JTAB-Registers
BDF0/1	Kopie des Interpreter-SEG-Registers
BDF2/3	Zeiger auf unteres Ende des Programm-Stacks (höchste freie Speicheradresse)
BDF4/5	IPC-Kopie beim Auftreten von Laufzeitfehlern
BDF6/7	Breakpoint-Zeilennummer
BDF8-BDFF	Breakpoint-Array für Diagnoseprogramm (Debugger) (ARRAY [0..3] OF INTEGER)
BE1C-BE3D	Konfigurationsparameter aus MISCINFO
BE1C	Lead-in-Zeichen für Bildschirm
BE1D	Cursor-Home-Zeichen
BE1E	Zeichen zum Löschen bis Bildschirmende
BE20	Cursor nach rechts bewegen
BE21	Cursor aufwärts bewegen
BE22	Backspace-Zeichen (Linkspfeil)

BE23	Verzögerung für Vertikalbewegungen
BE24	Zeilenlöschzeichen
BE25	Bildschirmlöschzeichen
BE28/9	Bildschirmhöhe (Zeilenanzahl)
BE2A/B	Bildschirmbreite (Spaltenzahl)
BE2C	Taste für Cursorbewegung nach oben
BE2D	Taste für Cursorbewegung nach unten
BE2E	Taste für Cursorbewegung nach links
BE2F	Taste für Cursorbewegung nach rechts
BE30	Taste für Dateiabschluß
BE31	Flush-Buffer-Zeichen (Sperrung der Tastatureingabe)
BE32	Break-Taste
BE33	Stop-Taste (Eingabestop)
BE34	Taste zum Löschen eines Zeichens
BE35	Nicht ausgebbares Zeichen
BE36	Taste zum Löschen einer Zeile
BE37	Editor-Escape-Taste
BE38	Lead-in-Zeichen für Tastatur
BE3A	Backspace (Cursor ein Zeichen nach links)
BE3E-BEFD	Segmenttabelle (ARRAY [0..31] OF RECORD UNITNUM : INTEGER; BLOCKNUM : INTEGER; CODELENG : INTEGER END)
BF00-BFFF	Variablen für Interpreter und BIOS
BF0A-BF0D	CONCKVECTOR (Zeiger auf Tastatur-Prüfungsroutine)
BF0E	SCRMODE (Wenn Bit 2 gesetzt -» Externes Terminal)
BF0F	LFFLAG (Bit 7 gelöscht -» Linefeeds auf Drucker schicken)
BF11	NLEFT (wird für horizontalen Bildschirmscroll verwendet)
BF12	ESCNT (Zähler für ESCAPE-Sequenzen)
BF13/4	RANDL & RANDH (Startwerte für Zufallsgenerator)
BF15	CONFLGS (Autofollow Bit = 0, Flush Bit = 6, Stop Bit = 7)
BF16/7	BREAK (Zeiger auf User-Break-Routine)
BF18	RPTR (Zeiger auf Terminal-Leseroutine)
BF19	WPTR (Zeiger auf Terminal-Schreibpuffer)
BF1A/B	RETL & RETH (Interpreter-Rücksprungadresse für BIOS-Aufrufe)
BF1C	SPCHAR (kontrolliert die Prüfung von Steuerzeichen) Bit 0 gesetzt -» keine Abfrage auf Ctrl A,Z,K,W, u. E; Bit 1 gesetzt -» keine Abfrage auf Ctrl-S und F
BF1D/E	IBREAK (Zeiger auf User-Break-Routine)
BF1F/20	ISYSCOM (Zeiger auf SYSCOM)
BF21	VERSION (02 = APPLE 1.1, 00 = APPLE 1.0)
BF22	FLAVOR (01 = reguläres System)

BF27-BF2E	SLTTYPS (Slot-Typentabelle)
BF2F/30	XITLOC Zeiger auf XIT-Befehl
BF56/BF7F	Wird vom FORTRAN-Kopierschutz verwendet
BFC0/BF7F	Für Fremdgeräte verfügbar
C000-CFFF	E/A-Adressen
D000-DFFF	BIOS-Code
D028	Disketten-Schreibroutine
D02C	Disketten-Leseroutine
D683	Disketten-Initialisierungsroutine
D69E	Kaltstart-Initialisierungsroutine
D772	Tastaturabfrage
D898	Terminal-Initialisierungsroutine
D8C6	Tastatur-Leseroutine
D8EF	Drucker-Initialisierungsroutine
D907	Initialisierungsroutine für Firmware Card
D918	Grafik-Initialisierungsroutine
D91C	Initialisierungsroutine für Fernanschluß
D923	Initialisierungsroutine für Communications Card
D930	Initialisierungsroutine für serielle Schnittstelle
D950	Bildschirm-Schreibroutine
D97B	Schreibroutine für Firmware Card
D98A	Schreibroutine für Fernanschluß
D9A4	Druckerinterface-Schreibroutine
D9B2	Schreibroutine für Communications Card
D9C3	Schreibroutine für Drucker
D9E5	Leseroutine für Fernanschluß
D9F8	Leseroutine für Communications Card
DA07	Leseroutine für Firmware Card
DA15	Leseroutine für serielle Schnittstelle
DCA0	Routinen für Fernanschluß- und Druckerstatus
DCB0	Bildschirm-Statusroutine
DCC5	Diskettenstatusroutine
DCE4	IDSEARCH-Routine
DDF5-DD28	Indextabelle (erster Buchstabe) für IDSEARCH
DE29-DFF9	Identifikator-Tabelle für IDSEARCH
D000-F22D	Interpreter Code
D000-D0FF	Hauptsprungtabelle für Interpreter
D100-D151	Sprungtabelle für den Aufruf von Standardprozeduren (CSP)
D253	Interpreter-Hauptprogramm
D25F	FJP (FALSE-Sprung)
D267	UJP (Unbedingter Sprung)
D296	LDCN (Lade Konstante NIL)
D29D	LDCI (Lade Ein-Wort-Konstante)

D2A9	SLDL1..SLDL16 (Ein-Wort-Kurzladebefehl)
D2B6	LDL (Lade lokales Wort)
D2D4	LLA (Lade lokale Adresse)
D2FA	STL (Speichere lokales Wort)
D318	SLDO1..SLDO16 (Kurzladebefehl für lokales Wort)
D325	LDO (Lade globales Wort)
D343	LAO (Lade globale Adresse)
D369	SRO (Speichere globales Wort)
D387	LOD (Lade Hilfswort)
D3AD	LDA (Lade Hilfsadresse)
D3DB	STR (Speichere Hilfswort)
D401	LDE (Lade erweitertes Wort)
D426	STE (Speichere erweitertes Wort)
D44B	LAE (Lade erweiterte Adresse)
D467	SIND1..SIND7 (Ein-Byte-Index; lade Wort)
D46A	SIND0 (Lade Wort indirekt)
D47B	STO (Speichere Wort indirekt)
D495	LDC (Lade aus mehreren Worten bestehende Konstante)
D4C8	LDM (Lade mehrere Worte)
D4F6	STM (Speichere mehrere Worte)
D523	LDB (Lade Byte)
D53D	STB (Speichere Byte)
D557	MOV (Verschiebe Worte)
D56B	LAND (Logisches UND)
D57E	LOR (Logisches ODER)
D591	LNOT (Logisches NICHT)
D59E	XJP (Case-Sprung)
D62F	NEW (CSP 1)
D66B	MARK (CSP 32)
D682	RELEASE (CSP 33)
D6A0	XIT (Betriebssystem verlassen)
D6BB	ABI (Absolutwert einer Integerzahl)
D6D9	ADI (Addiere Integer)
D6F1	NGI (Negiere Integer)
D703	SBI (Subtrahiere Integer)
D742	MPI (Multipliziere Integer)
D789	SQI (Integerquadrat)
D839	DVI (Dividiere Integer)
D866	MODI (Modulo-Division von Integerzahlen)
D87E	CHK (Überprüfe Untermengengrenze)
D8CD	LPA (Lade gepacktes Array)
D8E5	LSA (Lade konstante Stringadresse)
D907	SAS (Stringzuweisung)

D948	IXS (Indiziere Stringarray)
D96B	IND (Statischer Index; lade Wort)
D987	INC (Inkrementiere Feldzeiger)
D99A	IXA (Indiziere Array)
D9D9	IXP (Indiziere gepacktes Array)
DA1C	LDP (Lade gepacktes Feld)
DA72	STP (Speichere in ein gepacktes Feld)
DB20	INT (Schnittmenge)
DB57	DIF (Mengendifferenz)
DB79	UNI (Vereinigungsmenge)
DBE5	ADJ (Mengenanpassung)
DC55	INN (Element auf Enthaltensein in Menge prüfen)
DCBA	SGS (Erzeuge Ein-Element-Menge)
DD92-DDD3	Maskentabelle für Operationen auf gepackten Feldern
DDD4	NEQ (Ungleich)
DDD8	GRT (Größer als)
DDDC	LES (Kleiner als)
DDE0	GEQ (Größer oder gleich)
DDE4	LEQ (Kleiner oder gleich)
DDE8	EQU (Gleich)
DF2B	LESI (Integer kleiner als)
DF2F	GRTI (Integer größer als)
DF33	LEQI (Integer kleiner oder gleich)
DF37	GEQI (Integer größer oder gleich)
DF3B	NEQI (Integer ungleich)
DF65	EQUI (Integer gleich)
E253	CIP (Aufruf von Hilfsprozedur)
E2A1	CLP (Lokaler Prozeduraufruf)
E2BD	CGP (Globaler Prozeduraufruf)
E2D4	CXP (Aufruf externer Prozedur)
E2F9	CBP (Aufruf einer Basisprozedur)
E32A	RBP (Rücksprung von Basisprozedur)
E33F	RNP (Rücksprung von Nicht-Basisprozedur)
E417	Segment-Leseroutine
E61C	Lade residentes Segment (CSP 21)
E626	Verlasse residentes Segment (CSP 22)
E630	CSP (Aufruf einer Standardprozedur)
E63A	IDSEARCH (CSP 7)
E640	TREESEARCH (CSP 8)
E6B2	FILLCHAR (CSP 10)
E6F7	SCAN (CSP 11)
E784	EXIT (CSP 4)
E82B	BPT (Breakpoint)

E833	HALT (CSP 39)
E841	TIME (CSP 9)
E8A0	MOVELEFT (CSP 2) u. MOVERIGHT (CSP 3)
E940	MEMAVAIL (CSP 40)
EA2C	ADR (Addiere Real-Zahlen)
EB09	SBR (Subtrahiere Real-Zahlen)
EB5A	DVR (Dividiere Real-Zahlen)
EC55	MPR (Multipliziere Real-Zahlen)
EC7D	SQR (Quadriere Real-Zahl)
ECB2	ABR (Absolutwert einer Real-Zahl)
ECC0	NGR (Negiere Real-Zahl)
ED3F	FLO (Übertrage nächsten Wert auf Stackanfang)
ED62	FLT (Hole nächsten Wert vom Stackanfang)
EDBB	ROUND (CSP 24)
EDD0	TRUNC (CSP 23)
EDE5	PWROFTEN (CSP 36)
EE0E-EEA9	Zehnerpotenzen-Tabelle
EEAD-EEBC	Sprungtabelle zur Zeichenübertragung auf externe Geräte
EEBD-EECC	Sprungtabelle zum Lesen von Zeichen von externen Geräten
EECD	Prüfung, ob Gerät zulässig
EEF9	IORESULT (CSP 34)
EF04	IOCHECK (CSP 0)
EF0F	UNITBUSY (CSP 35)
EF1D	UNITWAIT (CSP 37)
EF27	UNITSTATUS (CSP 12)
EFA5	UNITCLEAR (CSP 38)
F069	UNITREAD (CSP 5)
F06E	UNITWRITE (CSP 6)
F22E-FE7B	Codesegment Nr.0, Prozeduren 1-28 von SYSTEM.PASCAL
FE80-FEAF	Sprungvektor für Benutzergerät
FEB0-FEC7	Disknummernvektor
FF00-FF41	BIOS-Sprungvektor vor Fold-Operation
FF5C-FF9D	BIOS-Sprungvektor nach Fold-Operation
FF5C-FF5E	Zeiger auf Tastatur-Leseroutine
FF5F-FF61	Zeiger auf Konsolen-Schreibroutine
FF62-FF64	Zeiger auf Terminal-Initialisierungsroutine
FF65-FF67	Zeiger auf Drucker-Schreibroutine
FF68-FF6A	Zeiger auf Drucker-Initialisierungsroutine
FF6B-FF6D	Zeiger auf Disketten-Schreibroutine
FF6E-FF70	Zeiger auf Disketten-Leseroutine
FF71-FF73	Zeiger auf Disketten-Initialisierungsroutine
FF74-FF76	Zeiger auf Fernanschluß-Leseroutine
FF77-FF79	Zeiger auf Fernanschluß-Schreibroutine

FF7A-FF7C	Zeiger auf Fernanschluß-Initialisierungsroutine
FF7D-FF7F	Zeiger auf Grafik-Schreibroutine
FF80-FF82	Zeiger auf Grafik-Initialisierungsroutine
FF83-FF85	Zeiger auf Drucker-Leseroutine
FF86-FF88	Zeiger auf Konsolen-Statusroutine
FF89-FF8B	Zeiger auf Drucker-Statusroutine
FF8C-FF8E	Zeiger auf Disketten-Statusroutine
FF8F-FF91	Zeiger auf Fernanschluß-Statusroutine
FF92-FF94	Zeiger auf Tastatur-Abfrageroutine
FF95-FF97	Zeiger auf eine Routine zum Ansprung eines Treiberprogramms über eine Sprungtabelle für Benutzergeräte
FF98-FF9A	Zeiger auf eine Routine zum Ansprung eines Treiberprogramms über den Diskettennummernvektor
FF9B-FF9D	Zeiger auf auf IDSEARCH
FFF6/7	Versionswort (0 = APPLE 1.1, 1 = APPLE 1.0)
FFF8-FFFF	Vektoren
FFF8/9	Startvektor
FFFA/B	Nicht maskierbarer Interrupt-Vektor
FFFC/D	RESET-Vektor
FFFE/F	Interrupt Request und BRK-Vektor

Mike Rosing/Keith McLauren

Pascal Intern

Einführung

Das UCSD-Pascal-System, wie es von der Firma APPLE Computer Inc. für den APPLE verwendet und modifiziert wurde, bietet dem Benutzer sehr komfortable Möglichkeiten zur Programmierung von 6502-Mikrocomputern. Leider ist die gemeinsame Verwendung von Maschinensprache und Pascal auf dem APPLE II nicht so bequem wie die gleichzeitige Nutzung von BASIC und Maschinensprache.

Die folgenden Fakten wurden nach der Methode "Versuch und Irrtum" gefunden. Wir hoffen daher, daß die Computer-Freaks, die ihren Computer so genau wie möglich kennen lernen wollen, eventuelle Fehler in diesem Artikel finden und berichtigen werden.

Für jemanden, der in seiner Freizeit den Versuch unternimmt, sämtliche Funktionen des Pascal-Betriebssystems in seiner ganzen Komplexität auszuforschen, wird daran sicherlich verzweifeln. Die folgenden Informationen sind daher hauptsächlich für die Assembler-Programmierer gedacht, die ihr Pascal-Betriebssystem besser unter Kontrolle haben wollen.

In Teil 1 wird der Pascal-Bootvorgang im einzelnen erklärt. Er ist durchaus nicht trivial. Allein durch das Verfolgen des Bootvorganges kann man eine Menge über den APPLE II lernen.

Teil 2 enthält eine detaillierte Beschreibung, wie die Pascal-E/A-Routinen zu verwenden sind.

In Teil 3 wird das Pascal-Disketteninhaltsverzeichnis (Directory) erklärt. Diese Beschreibung ist insbesondere im Zusammenhang mit Teil zwei nützlich, wenn man Programme speichern will.

Teil 4 erklärt einfach, wie der 6502-Maschinencode zur Interpretation von P-Code-Instruktionen verwendet wird. In den darauf folgenden Tabellen werden die Einsprungadressen für die P-Code-Instruktionen aufgelistet.

In Teil 5 wird beschrieben, wie die Speichernutzung unter Pascal-erfolgt. Dabei wird auf Abweichungen der Pascal-Handbücher und der tatsächlichen Implementierung hingewiesen.

Anhang 1 enthält die Adressen der P-Maschinen-Register. Zum Ver-

ständnis dieser Register benötigt man die Pascal-Handbücher, die mit dem Betriebssystem verkauft werden.

Wer sich ernsthaft für APPLE Pascal interessiert, sollte sich BIOS besorgen, das *BASIC Input Output System*, das von APPLE veröffentlicht wurde. Die neueste Version heißt SYSTEM.ATTACH und kann bei *International APPLE Core*, 910-A George St., Santa Clara, CA 95050 für $7.50 erworben werden.

Innerhalb dieses Artikels wird Version 1.1 als die "neue Version" und das Original-APPLE-Pascal als "alte Version" bezeichnet.

Booten

Der Bootvorgang des APPLE-Pascal erfolgt in vier Stufen. Das hängt damit zusammen, daß ein Teil des Bootens durch Pascal-Routinen erfolgt (vgl. auch *APPLE Orchard*, 1980: *APPLE DOS Booting Process* von Ted Burns). Die erste Stufe erfolgt durch das ROM; sie ist damit offensichtlich die gleiche wie bei BASIC-Disketten. In der zweiten Stufe wird SYSTEM.PASCAL in die 16k-Language Card geladen. In der dritten Stufe wird das BIOS geladen, und die P-Code-Zeiger werden initialisiert. Dabei wird der Interpreter aktiviert und die vierte Bootstufe in Pascal-P-Code ausgeführt.

Disassembliert man die Blöcke 0 und 1, so findet man folgende Anweisung:

```
08A9 6C F8 FF JMP §FFF8
```

Mit 08A9 wird die zugehörige Speicherzelle (hexadezimal) bezeichnet, 6C F8 FF ist der Maschinencode und JMP §FFF8 die entsprechende, von Menschen lesbare Assembleranweisung. Die Syntax wird in den Pascal-Handbüchern in dem Kapitel *Assembler* beschrieben.

Mit dieser Sprunganweisung beginnt die dritte Stufe des Bootvorganges. Die angesprungene Adresse ist FEE9 (neue Version) bzw. FF65 (alte Version). An dieser Stelle wird das BIOS geladen und das Pascal-Betriebssystem in die Language Card kopiert.

Die dritte Bootstufe, mit der die vierte Stufe geladen wird, ist in Maschinensprache geschrieben. Die vierte Stufe, die aus P-Code besteht, überschreibt Teile des von Stufe 3 benutzten Speicherbereiches. In der alten Pascal-Version blieb die dritte Stufe im Hauptspeicher, so daß ein Warm-Restart möglich war; die neue Version kann dagegen nur Kaltstarts ausführen, da die dritte Bootstufe wichtige Zeiger initialisiert.

Die vierte Stufe ist in Pascal geschrieben und hat in der

SYSTEM.PASCAL-Datei keinen eigenen Namen. Sie ist das Segment Nr.15 der SYSTEM.PASCAL-Datei.

Hier nun eine detaillierte Beschreibung des Bootprozesses. Sie kann möglicherweise Fehler enthalten - aber das müßten Sie dann selbst überprüfen.

1) Das ROM-Programm springt zu Adresse C600 im ROM und liest Spur 0, Sektor 0 bei $800 beginnend ein.
2) Sprung nach $801 und Nachladen des Rests von Bootstufe Nr. 2.
3) Prüfung, ob SYSTEM.APPLE in Block zwei (Start des Directories) zu finden ist; falls nicht, wird die Meldung NO FILE SYSTEM APPLE auf dem Schirm ausgegeben und folgende Anweisung ausgeführt, die eine hübsche kleine Endlosschleife darstellt:

   0869 F0 FE BEQ 869

4) Wird SYSTEM.APPLE im Directory gefunden, so wird ein Write-Enable für das RAM ausgeführt. Das BIOS wird in die zweite 4k-Bank und E000 bis FFFF geladen. Dann wird die zweite 4k-Bank aktiviert und der P-Code-Interpreter geladen (wegen der Bank-Umschaltung vgl. das *Language-Card-Handbuch*).
5) Für die Language Card wird ein Read-Write-Enable ausgeführt, die zweite 4k-Bank angeschaltet und die BIOS-Reset-Adresse D69E (neu) bzw. D5DD (alt) angesprungen.
6) Die Speicheradressen 0 bis BFFF werden gelöscht.
7) Alle Slots werden auf ROMs überprüft. Das Ergebnis dieses Tests wird bei der neuen Version in Adresse BF27, bei der alten Version bei BFF8 abgelegt. Die Ergebniscodes lauten:

   0 kein ROM
   1 nicht identifizierbares ROM
   2 Disketten-Karte
   3 Communications Card
   4 serielle Schnittstelle
   5 Drucker
   6 Smart-Term-80-Zeichen-Karte (nur neue Version)

   Die Adresse plus Slotnummer ergibt den zugehörigen Code.
   Beispiel:

   BF2D = 02 neue Version
   BFFE = 02 alte Version

8) Der Bildschirm wird gelöscht und der Cursor auf den Bildschirm gesetzt; ggf. vorhandene externe Geräte und die Disk-Laufwerke werden initialisiert. In der neuen Version werden noch zusätzliche, kleinere Operationen ausgeführt.
9) Sprung nach F275 (neu) bzw. EF3F (alt). Bei der neuen Version wird der Bereich F275 bis F675 nach 6800 bis 6C00 verschoben. Dieser Teil wird dadurch überschrieben.
10) Löschen des Stacks. Bei der neuen Version wird der Reset-Vektor auf D6A0 gesetzt. SYSCOM wird auf BDDE gesetzt und der Bereich BDDE bis BEFE (neue Version) bzw. BDDE bis BEDE (alte Version) gelöscht. Prüft überflüssigerweise, ob Unit #4 (Boot-Laufwerk) angeschlossen ist.
11) Einlesen des Directories ab Adresse $6000. Danach wird nach SYSTEM.PASCAL gesucht (bis zu 78 mal). Wird dieser Systemteil nicht gefunden, so wird die Konsole initialisiert und folgendes ausgegeben:

    alte Version: nichts
    neue Version: Insert boot disk with SYSTEM.
    PASCAL on it, then press RESET

    Die neue Version ändert die RESET-Vektoren. Danach führen beide Versionen folgendes aus:

    D0FE BNE 696E (neu)
    D0FE BNE EF94 (alt)

    und warten sehr geduldig auf RESET.
    Nach dem RESET wird ein Sprung an die Spitze des BIOS ausgeführt (Schritt 5) und die ganze Sache wird wiederholt.

12) Wird SYSTEM.PASCAL im Directory gefunden, so wird der erste Block (Block 0) ab Adresse $6000 eingelesen.
13) Eine unbenannte Datei mit einer Länge von $1082 Bytes wird in die Adressen AC9A bis BD1B eingelesen (neue Version). Für die alte Version gilt: Länge: $1258, Adressen ABC3 bis BD1B. Es handelt sich dabei um Segment Nr.F (15).
14) Segment Nr. 0 von SYSTEM.PASCAL wird in die Adressen F22E bis FE7C kopiert (alte Version: F104 bis FEFC).
15) Der erste Activation Record für PASCALSY wird erzeugt (Segment Nr. 0) und der IPC auf F22E (neu) bzw. F104 (alt) gesetzt.
16) P-Code-Sprungbefehle werden in 6E bis 73 eingelesen.
17) Aufruf des P-Code Decoders bei D253 (neu) bzw. D243 (alt).

18) Vierte Bootstufe ist PASCALSY. Von hier an werden P-Code-Instruktionen ausgeführt.

Die Benutzung von Disketten-E/A

Vor einem Schreib- oder Lesezugriff auf eine Diskette müssen bei der alten Version acht Elemente auf den Stack geschoben (push) werden. Die neue Version erfordert hierfür zehn Elemente.

Grundsätzlich gibt es zwei verschiedene Möglichkeiten für den Zugriff auf die Disketten-E/A-Routinen. Die erste verwendet die von APPLE in BIOS vorgeschlagenen Sprunganweisungen. Diese Methode funktioniert sowohl mit der alten als auch mit der neuen Pascal-Version, sie erfordert lediglich ein paar zusätzliche Push-Operationen auf dem Stack. Die zweite Zugriffsmethode besteht darin, die zweite 4k-Bank des BIOS zu aktivieren und die Diskettenroutinen direkt anzusprechen.

Dabei müssen folgende Parameter auf dem Stack abgelegt werden: Unitnummer, Blocknummer, Pufferbereich, Anzahl der zu lesenden oder zu schreibenden Sektoren oder Bytes und schließlich der Stack Marker. Die alte Version benutzt nur einen Stack Marker, die neue Version dagegen drei. Die Reihenfolge der Parameter auf dem Stack ist:

1) Stack Marker (einer bei der alten, drei bei der neuen Version)
2) Unitnummer
3) Pufferbereich
   a) Hi-Byte
   b) Lo-Byte
4) Anzahl der Bytes
   a) Hi-Byte
   b) Lo-Byte
5) Blocknummer auf der Diskette
   a) Hi-Byte
   b) Lo-Byte

Hier ein Beispiel für das Speichern eines FOTO-Files auf der Diskette:

```
         .PROC SAVEFOTO
         JMP START
;Fotofl ist der Anfang von HI RES Seite 1
;Lngth ist die Laenge in Bytes der zu
;speichernden Datei
;Unit zeigt auf #4 (Laufwerk 1)
;Blknm zeigt auf Block 6, es werden
;mindestens 8 Blocks benoetigt, um
;8 K Arbeitsspeicher aufzunehmen

FOTOFL   .WORD   02000
LNGTH    .WORD   01FFF
UNIT     .BYTE   00
BLKNM    .WORD   0006

START    LDA     #00          ;Stack Marker
         PHA                  ;fuer
         PHA                  ;neue Version
         PHA                  ;
         LDA     UNIT         ;Laufwerk, auf das
         PHA                  ;geschrieben wird
         LDA     FOTOFL+1     ;High Byte
         PHA                  ;der Pufferadresse
         LDA     FOTOFL       ;zuerst
         PHA                  ;
         LDA     LNGTH+1      ;Ebenso bei
         PHA                  ;Puffergroesse
         LDA     LNGTH        ;
         PHA
         LDA     BLKNM+1      ;Auf Diskette
         PHA                  ;an sicherem
         LDA     BLKNM        ;Platz ablegen
         PHA                  ;
         JSR     OFFOF        ;E/A-Aufruf
         RTS
```

Alternativ dazu kann man diese Parameter auf dem Stack ablegen und folgende Befehle ausführen:

```
LDA 0C083 ;BIOS einschalten
JSR 0D028 ;neue Version
LDA 0C08B ;Zurückschalten auf P-Code
```

Bei der alten Version muß hier D038 eingesetzt werden.

Für das Lesen von Disketten werden die gleichen Parameter auf dem Stack abgelegt. Die JSRs müssen leicht geändert werden. Denken Sie daran, daß das Directory unverändert bleibt, wenn Sie diese Operation ausführen. Seien Sie daher nicht überrascht, wenn das Pascal so gespeicherte Daten wieder überschreibt. Wie das Directory aufgebaut ist, werden wir im nächsten Abschnitt besprechen.

Diskette lesen FF12, D02C (neu)
Diskette schreiben FF0F, D028
Diskette lesen FF12, D03C (alt)
Diskette schreiben FF0F, D038

Die D0XX-Adressen beziehen sich auf die zweite 4k-Bank. Der Aufruf der FFXX-Adressen schaltet die zweite 4k-Bank ein, erledigt die E/A-Operationen und schaltet automatisch auf die erste Bank zurück.

Das PASCAL-Directory

Im Gegensatz zu BASIC, das Disketten in Spuren und Sektoren einteilt, verwendet das UCSD-Pascal für Disketten eine Blockstruktur. Die von DOS 3.3 verwendeten 16-Sektor-Disketten sind bis auf ein paar Einschränkungen mit Pascal kompatibel. Die beiden unterschiedlichen Sprachen verwenden das gleiche Diskettenformat, und Disketten, die von dem einen Betriebssystem formatiert wurden, können von dem anderen gelesen werden. Ein Beispiel dafür ist das DOS 3.3 COPY-Programm, das zum Teil aus BASIC-Befehlen und zum Teil aus Maschinensprache(d.h. Assembler)-Routinen besteht. Mit diesem Programm lassen sich auch Pascal-Disketten mit einem oder zwei Laufwerken kopieren, ohne daß man dazu eine formatierte Diskette braucht. Es gibt noch einige andere BASIC-Hilfsprogramme, die sich ebenfalls auf Pascal-Disketten anwenden lassen.

Das Hauptproblem beim Lesen von BASIC-Disketten unter Pascal (und umgekehrt) besteht darin, daß man für jeden Block, auf den zugegriffen werden soll, eine Umrechnung in die entsprechende Spur- und Sektornum-

mer vornehmen muß. Die Spurnummer erhält man durch Division durch acht, während man die Sektornummer in der nachstehenden Tabelle nachsehen kann (s. Sektorzuordnung für APPLE-Pascal).

Das Directory ist bei Pascal-Disketten von besonderem Interesse, da man, wenn man dieses Inhaltsverzeichnis kennt, Programme und Daten von defekten Disketten "retten" kann. Hier zunächst einige sehr allgemeine Informationen. Die Diskette besteht aus 35 Spuren zu je 16 Sektoren. In Pascal werden diese Sektoren zu Zweiergruppen, den sog. Blöcken, zusammengefaßt. Untersucht man das Directory vom BASIC aus, so erhält man einige Hinweise, die man für die Rettung von Daten gut gebrauchen kann. Das Directory beginnt in Spur 0, Sektor 11 und setzt sich nach unten fort. Der nutzbare Bereich befindet sich in den Sektoren 11 bis 2 (Blöcke 2 bis 5). Die Sektoren 0,1,14,13 und 12 werden vom Betriebssystem zum Booten verwendet und sind nur für Disketten wichtig, die SYSTEM.APPLE enthalten. Die verbleibenden Blöcke 6 und 7 werden von Pascal zum Speichern von Dateinamen verwendet.

Die von Pascal verwendete Sektornummerierung ist fast linear rückwärts, mit Ausnahme der Blöcke 0 und 7 als erster und letzter Block einer Spur. Unter APPLE-Pascal besteht jede Spur aus acht Blöcken, denen die Sektornummern der Tabelle entsprechend zuzuordnen sind. Dieses Zuordnungsmuster ist für alle Spuren gleich.
Das Directory beginnt bei Block Nr.2, d.h. Spur 0, Sektor 11 und enthält als ersten Eintrag den Namen der Diskette und das Datum.

Auf den Diskettennamen folgen nacheinander die Dateinamen. Bei allen Datei- oder Diskettennamen steht in der vorangehenden Stelle eine Hexadezimalzahl, die deren Länge angibt. Das Datum wird mit zwei Bytes codiert (Nr.20 und 21 werden vom Filer für das aktuelle Datum verwendet). Die Bytes 25 und 26 (vom Anfang jedes Directory-Eintrages aus gezählt) enthalten das Erstellungsdatum des jeweiligen Files. Das Datum besteht aus 16 Datenbits, die sich aus der Verknüpfung der Bytes 26 und 25 ergeben. Das Jahr ist in den Bits 7 bis 1 von Byte 26 und der Tag in Bit 0 von Byte 26 und den Bits 7 bis 4 in Byte 25 untergebracht. Der Monat schließlich setzt sich aus den Bits 3 bis 0 aus Byte 25 zusammen. Die Datum-Informationen bestehen aus Binärzahlen, und die Monate werden durch die Zahlen 0 bis 12 dargestellt.

Die Bytes 1 und 2 eines Directory-Eintrages enthalten die Block-Startnummer, und die Bytes 3 und 4 enthalten die Block-Endnummer der zugehörigen Datei. In den Bytes 5 und 6 ist der Dateityp verschlüsselt. Die Zahlen 2, 3 und 5 geben zum Beispiel an, ob es sich um ein Code-, Text- oder Datenfile handelt. Das siebente Byte enthält die Länge des Dateinamens, während der Name selbst in den folgenden 15 Bytes untergebracht ist. Die Bytes 23 und 24 geben die Anzahl der im letzten Dateiblock vom File belegten Datenbytes an. Darauf folgt in den Bytes 25 und 26 das bereits beschriebene File-Datum. In allen Byte-Paaren wird das höherwertige Byte hinter das niederwertige Byte

gesetzt; eine "5" wird also durch das Byte-Paar "05 00" dargestellt. Der erste Dateieintrag beginnt bei Byte Nr.26 und besteht seinerseits aus 26 Bytes, der nächste Eintrag besteht aus den darauffolgenden 26 Bytes usw.

Dateien werden durch Vermindern der Dateianzahl und durch Komprimieren des Directories gelöscht. Befindet sich der entsprechende Dateieintrag am Ende der Dateiliste, kann das Directory nicht weiter komprimiert werden; deshalb wird in diesem Fall nur die Dateianzahl vermindert. Aus diesem Grund kann man auch Dateien wiederherstellen, wenn sie am Ende des Directories standen und die Disketteninformationen in der Zwischenzeit nicht durch Schreibzugriffe geändert wurden. Das Rekonstruieren von gelöschten Dateien ist allerdings nicht mehr so einfach, wenn der entsprechende Directory-Eintrag irgendwo in der Mitte der Dateiliste stand. In diesem Fall muß die Lage des Files auf der Diskette entweder durch die "E(xtended Listing"-Funktion des Filers oder durch direkte Untersuchung der einzelnen Diskettenblocks ermittelt werden.

Aufbau des Pascal-Dateiverzeichnisses

Byte-Nr.	
0-25	Diskettenname und zugehörige Informationen
26-51	Erster Dateieintrag
52-77	Zweiter Dateieintrag
MOD 26	Weitere Dateien

Disketteninformationen

1,0	Beginn des Directories
3,2	Nummer des letzten Directory-Blocks
5,4	? immer null ?
6	Länge des Diskettennamens
7-13	Diskettenname
15,14	Anzahl der Diskettenblocks
17-16	? immer null ?
19,18	? immer null ?
21,20	Aktuelles Datum
23,22	? immer null ?
24-26	? immer null ?

Einzelne Dateien

1,0	Nummer des Anfangsblocks
3,2	Nummer des Endblocks
5,4	Dateityp: 2 = Code, 3 = Text, 5 = Data
6	Länge des Dateinamens (Maximal 15)
7-21	Dateiname
23,22	Anzahl der Bytes im letzten Block
25,24	Dateidatum

Datum des Files

Byte 25		Byte 24
(7654321)	(07654)	(3210)
Jahr	Tag	Monat
0..99	0..31	1..12

Sektorzuordnung für APPLE-Pascal

Block-Nummer	**Sektor-Nummer**
0	0,14
1	13,12
2	11,10
3	9, 8
4	7, 6
5	5, 4
6	3, 2
7	1,15
(nächste Spur)	
8	0,14
	usw.

wenigstens 34 Mal wiederholen

Mit Hilfe von Record-Konstrukten läßt sich die Directory-Struktur als Pascal-Datentyp beschreiben. Im *Byte* vom Mai 1981 ist ein Programm veröffentlicht, das diese Information benutzt, um Pascal-Disketten zu katalogisieren.

Die Funktionsweise des P-Code-Interpreters

Hier nun unsere Interpretation, wie Pascal die P-Codes auf dem 6502-Mikroprozessor ausführt. Ich bewundere die Leute, die dieses Programm geschrieben haben; der Code ist sehr kompakt und effizient.

Die erste Programmzeile ist bei der alten Version D243 und bei der neuen Version D253.

```
LDY #00
LDA §58,Y     ; Hole P-Code.
BPL PUSH      ; Wenn kein P-Code, lege Wort auf Stack ab.
ASL A         ; Code mal zwei.
STA 6F        ; Änderung der indirekten Sprungadresse.
JMP 006E      ; Springe zu Einsprungadresse.
```

Bei den Adressen 58 und 59 liegt der sog. IPC, der *Interpreter Program Counter*. Dieser Wert ist ein Zeiger, der auf das aktuell zu verarbeitende Byte weist. Ist die dort stehende Zahl kleiner als 7F, dann wird sie und das darauffolgende Byte auf den System-Stack übertragen. Hat die Zahl einen größeren Wert, so wird sie mit zwei multipliziert. Der Grund hierfür ist, daß man für einen 16-Bit-Zeiger zwei Bytes benötigt. Das Resultat wird in der Zero-Page bei Adresse 6F gespeichert, die als Lo-Byte für einen indirekten Sprungbefehl verwendet wird. Dieser Sprungbefehl steht auf der Zero-Page als:

```
006E 6C XX D0 JMP §D0XX
```

Der Wert XX entspricht dabei dem Lo-Byte des Befehls, während D0 die Seite der Lookup-Tabelle darstellt. Also ist der P-Code nur ein Index für eine Sprungtabelle, mit deren Hilfe der P-Code interpretiert wird. Der 6502-Rechner benötigt dabei für eine P-Code-Instruktion nur 24 Mikrosekunden!

In Tabelle 1 sind die P-Code-Namen und ihre Adressen aufgelistet. Die P-Codes selbst werden in Ihren APPLE-UCSD-Handbüchern beschrieben.

Zusätzlich zu diesen Op-Codes gibt es einige Spezialprozeduren (CSP = 9E). Sie werden in Tabelle 2 aufgelistet. Es sind nicht sehr viele, und sie haben keine eigenen Namen. In der neuen Version gibt es eine weitere Prozedur bei 9E 0C. Ich überlasse es Ihnen, herauszufinden, wozu diese Prozedur benutzt wird.

```
type      date_record   = packed record        (* auf 16 Bits verteilt: *)
                            month : 1..12      (* vier Bits fuer Monat  *)
                            day   : 1..31      (* fuenf Bits fuer Tag   *)
                            year  : 0..99      (* sieben Bits fuer Jahr *)
          end;

          vol_id         = string [7]          (* Diskettenname z.B. APPLE3 *)
          file_id        = string [15]            (* Filename im Directory*)

          (* Die Dateitypen werden aus einer nummerierten Liste der neun
          UCSD-Dateitypen abgeleitet. Bis auf die Typen info, graf, foto
          und secure unterstuetzt APPLE-Pascal alle diese Dateitypen. Der
          Filer kann alle Typen unterscheiden, und der Benutzer kann daher
          eigene info-, foto- und graf-Files implementieren. Man darf dabei
          jedoch nicht erwarten, da  die zukuenftigen Pascal-Implementierun-
          gen mit den selbstimplementierten Dateitypen (z.B. FOTO-Dateien)
          kompatibel sind.*)

          file_type     = (untyped, badblk, code, text, info, data, graf,
                           foto, securedir);
          (* Hier beginnt erst der eigentliche Directory-Record, waehrend
          obige Konstruktion das Lesen erleichtert. *)

          dir_record    = record
                            first_block    integer;
                            last_block     integer;

          (* die beiden unterschiedlichen Typen von Directory-Eintraegen
          werden durch die CASE-Anweisung ausgedrueckt *)

                            case dir_file_kind : file_type of
                            securedir, untyped : (* erste Variante *)
                              (dir_vol_name  : vol_id;
                               zero_blocks, num_of_files, total_blocks:integer;
                               last_boot : date_record);
                            badblk, code, text, info, data, graf, foto :
                                                          (* zweite Variante *)
                              (dir_file_name : file_id;
                              last_byte      : 1..512;
                              dir_file_date : date_record);
          end;
```

Tab. 1

Decimal	Hex	Mnemonic	Location old	Location new
128	80	ABI	D698	D6BB
129	81	ABR	EAF2	ECB2
130	82	ADI	D6B6	D6D9
131	83	ADR	E902	EAC2
132	84	AND	D548	D56B
133	85	DIF	DB2C	DB57
134	86	DVI	D815	D839
135	87	DVR	E99A	EB5A
136	88	CHK	D85A	D87E
137	89	FLO	EB6F	ED3F
138	8A	FLT	EB92	ED62
139	8B	INN	DC2A	DC55
140	8C	INT	DAF5	DB20
141	8D	IOR	D55B	D57E
142	8E	MOD	D842	D866
143	8F	MPI	D71F	D742
144	90	MPR	EA95	EC55
145	91	NGI	D6CE	D6F1
146	92	NGR	EB00	ECC0
147	93	NOT	D56E	D591
148	94	SRS	DCA1	DCCC
149	95	SBI	D6E0	D703
150	96	SBR	E949	EB09
151	97	SGS	DC8F	DCBA
152	98	SQI	D766	D789
153	99	SQR	EABD	EC7D
154	9A	STO	D46B	D47B
155	9B	IXS	D924	D948
156	9C	UNI	DB4E	DB79
157	9D	S2P	D3F1	D401
158	9E	CSP	E5B1	E630
159	9F	LDCN	D286	D296
160	A0	ADJ	DBBA	DBE5
161	A1	FJP	D24F	D25F
162	A2	INC	D963	D987
163	A3	IND	D947	D96B
164	A4	IXA	D976	D99A
165	A5	LAO	D333	D343
166	A6	LSA	D8C1	D8E5
167	A7	LAE	D43B	D44B
168	A8	MOV	D534	D557
169	A9	LDO	D315	D325
170	AA	SAS	D8E3	D907
171	AB	SRO	D359	D369
172	AC	XJP	D57B	D59E
173	AD	RNP	E2FC	E33F
174	AE	CIP	E210	E253
175	AF	EQU	DDBD	DDE8
176	B0	GEQ	DDB5	DDE0
177	B1	GRT	DDAD	DDD8
178	B2	LDA	D39D	D3AD

Decimal	Hex	Mnemonic	Location	
			old	new
179	B3	LDC	D485	D495
180	B4	LEQ	DDB9	DDE4
181	B5	LES	DDB1	DDDC
182	B6	LOD	D377	D387
183	B7	NEQ	DDA9	DDD4
184	B8	STR	D3CB	D3DB
185	B9	UJP	D257	D267
186	BA	LDP	D958	DA1C
187	BB	STP	DA4E	DA72
188	BC	LDM	D4B8	D4C8
189	BD	STM	D4D3	D4F6
190	BE	LDB	D500	D523
191	BF	STB	D51A	D53D
192	C0	IXP	D9B5	D9D9
193	C1	RBP	E2E7	E32A
194	C2	CBP	E2B6	E2F9
195	C3	EQUI	DF3A	DF65
196	C4	GEQI	DF0C	DF37
197	C5	GRTI	DF04	DF2F
198	C6	LLA	D2C4	D2D4
199	C7	LDCI	D28D	D29D
200	C8	LEQI	DF08	DF33
201	C9	LESI	DF00	DF2B
202	CA	LDL	D2A6	D2B6
203	CB	NEQI	DF10	DF3B
204	CC	STL	D2EA	D2FA
205	CD	CXP	E291	E2D4
206	CE	CLP	E25E	E2A1
207	CF	CGP	E27A	E2BD
208	D0	LPA	D8A9	D8CD
209	D1	STE	D416	D426
210	D2	NOP	D23D	D24D
211	D3	EFJ	D212	D1EF
212	D4	NFJ	D212	D1EF
213	D5	BPT	E67E	E82B
214	D6	XIT	D67D	D6A0
215	D7	NOP	D23D	D24D
216	D8	SLDL1		
:	:	:	D299	D2A9
231	E7	SLDL16		
232	E8	SLDO1		
:	:	:	D308	D318
247	F7	SLDO16		
248	F8	SINDO	D45A	D46A
249	F9	SIND1		
:	:	:	D457	D467
255	FF	SIND7		

Die Pascal-Struktur

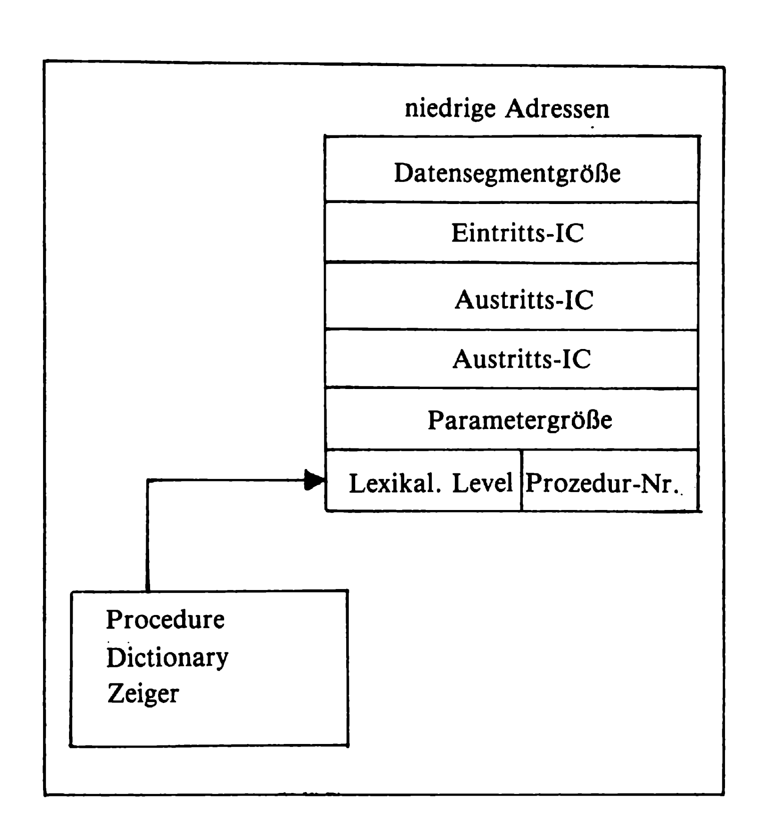
Nachdem ich die APPLE-Handbücher über das UCSD-Pascal-System ungefähr zehnmal gelesen hatte, war ich total durcheinander und wußte überhaupt nicht mehr weiter. Als ich jedoch anfing, den Maschinencode Schritt für Schritt nachzuverfolgen, klärten sich eine Menge Fragen. Die übrig gebliebenen Ungereimtheiten hängen im wesentlichen mit den Unterschieden zwischen dem UCSD-Pascal und der von APPLE implementierten Version zusammen.

Nachfolgend finden Sie eine Skizze der Speichernutzung für ein beliebiges Pascal-Programm. Die wesentlichen Bestandteile des Programms sind: das SEGMENT DISK DICTIONARY und das SEGMENT PROCEDURE DICTIONARY. Das Segment Disk Dictionary liegt im Bereich BE3E bis BE6D und enthält Angaben über das Laufwerk, die Block-Nummer und die Länge jedes Programmsegments. Das Segment Procedure Dictionary liegt bei BD9E bis BDBD und enthält selbst-relative Zeiger auf jede Procedure-Dictionary-Tabelle eines Segments.

Wozu braucht man diese Dictionaries? Wenn eine Prozedur mit dem P-Code aufgerufen wird, muß das Pascal-System diese Prozedur im Hauptspeicher finden. Dazu wird zunächst eine Sprungtabelle bei BD1E bis BD3D (Invocation Table) durchsucht, um festzustellen, ob sich das gewünschte Segment im Hauptspeicher befindet. Ist das nicht der Fall, so wird das Segment Disk Dictionary abgesucht, um das Segment auf der Diskette zu finden und in den Hauptspeicher zu laden. Befindet sich das Segment schon im Arbeitsspeicher, dann durchsucht das Pascal-Betriebssystem das Segment Procedure Dictionary nach dem Prozedurverzeichnis (Procedure Dictionary) des benötigten Segments.

Das Procedure Dictionary enthält seinerseits selbst-relative Zeiger auf den Markstack der jeweiligen Prozeduren. Durch die Verwendung derartiger selbst-relativer Zeiger ist das Pascal so flexibel. Im Procedure Dictionary sind folgende Parameter verzeichnet: die Anzahl der Prozeduren im Dictionary, die Segmentnummer und die Wortliste mit den selbst-relativen Zeigern auf alle Prozeduren eines Segments. Dies wird in den APPLE-Handbüchern auch richtig beschrieben.

Zu jeder Prozedur gehört eine Sprungtabelle, die oft als JTAB (Jump Table) bezeichnet wird. An dieser Stelle stimmt die Beschreibung der Handbücher mit dem, was tatsächlich implementiert wurde, nicht überein. Die folgende Zeichnung gibt den wirklichen Sachverhalt wieder:


niedrige Adressen
Datensegmentgröße
Eintritts-IC
Austritts-IC
Austritts-IC
Parametergröße
Lexikal. Level
Prozedur-Nr.
Procedure
Dictionary
Zeiger

Die Interpreterzeiger für Prozedureintritt und -austritt (IC = Interpreter Counter) sind selbst-relative Zeiger auf den Anfang bzw. das Ende des jeweiligen Prozedurcodes. Die Parametergröße muß kleiner als 255 sein, wogegen die Größe des Datensegments theoretisch beliebig ist.

Um zu sehen, wozu alle diese Angaben benötigt werden, wollen wir eine P-Code-Anweisung verfolgen, mit der eine Prozedur aufgerufen wird.

Das einfachste Beispiel ist CLP Nr.P, also "Call Local Procedure" (Aufruf einer lokalen Prozedur mit Nr.P). Dieser P-Code-Befehl speichert P zunächst in Zero-Page-Adresse Nr.78. Danach wird automatisch eine Unterroutine zum Aufbau eines Activation Records aufgerufen (BUILD AN ACTIVATION RECORD). Diese Prozedur liegt für externe Prozeduren bei E0A1 und für lokale Prozeduren bei E0BC.

Diese Routine ist wahrscheinlich eine der wichtigsten Pascal-Prozeduren, denn sie bestimmt den Markstack für jede einzelne Prozedur. Da das Pascal-Betriebssystem selbst in Pascal geschrieben ist, ruft jede Prozedur eine weitere Prozedur auf, bis schließlich eine der Kernprozeduren erreicht wird, die in Maschinensprache geschrieben sind.

Beim Abarbeiten der Prozedur wird auf die übergebenen Parameter und die Sprungtabelle im Activation Record zugegriffen und der neue Basis-Adreßzeiger an das untere Ende des neuen Stapels (Pile) gesetzt. Reicht der Stapel über den Anfang des Heaps hinaus, so tritt ein *Stack Overflow* auf.

Der dazu notwendige Code nimmt etwas weniger als einen Block Speicherplatz ein. Wenn man ihn näher untersucht, bekommt man einen plastischen Eindruck, wie effektiv Pascal tatsächlich ist.

Eine weitere wichtige Unterroutine ist der "Invocation Counter" (Aufrufzähler). Diese Prozedur beginnt bei E4A5. Wird ein neues Segment aufgerufen, das noch nie benutzt worden ist, liest diese Unterroutine sie von der Diskette ein und sucht in diesem Segment nach dem lexikalischen Level 0 und der Prozedur Nummer 0. Sind diese nicht vorhanden, so kann das zu einer Endlosschleife führen; in diesem Fall ist das Segment aber sowieso unbrauchbar.

Ist das Segment schon einmal aufgerufen worden, wird die Aufruftabelle an der entsprechenden Stelle um 1 vermindert, um anzuzeigen, daß das Segment noch verwendet wird, sowie, auf welchem Level es benutzt wird.

Im folgenden Anhang finden Sie eine Liste der benutzten Zero-Page-Zeiger und Tabellen. Da das Betriebssystem so komplex ist, können wir nicht garantieren, daß die Beschreibungen in dieser Tabelle vollständig richtig sind. Seien Sie daher vorsichtig, wenn Sie mit dem Markstack experimentieren!

Tab. 2

Spezielle Prozeduren (alle mit vorabstehendem 9E (CSP))

Decimal	Hex	Mnemonic	Location old	Location new
0	0		ED26	EF04
1	1	NEW	D60C	D62F
2	2	MVL	E6F3	E8A0
3	3	MVR	E6F3	E8A0
4	4	EXIT	E5D9	E784
5	5		EDD9	F069
6	6		EDDE	F06E
7	7	IDS	E5BB	E63A
8	8	TRS	E5CD	E640
9	9	TIM	E694	E841
10	A	FLC	E5C1	E6B2
11	B	SCN	E5C7	E6F7
12	C		----	EF27
13	D			
:	:	UNUSED	----	----
20	14			
21	15		E59D	E61C
22	16	TNC	E5A7	E626
23	17	RND	EC00	EDD0
24	18	SIN	EBEB	EDBB
25	19			
:	:	MATH	D212	D1EF
31	1F			
32	20	MRK	D648	D66B
33	21	RLS	D65F	D682
34	22		ED1B	EEF9
35	23	POT	ED31	EF0F
36	24		EC15	EDE5
37	25		ED3F	EF1D
38	26		ED49	EFA5
39	27		E686	E833
40	28		E757	E904

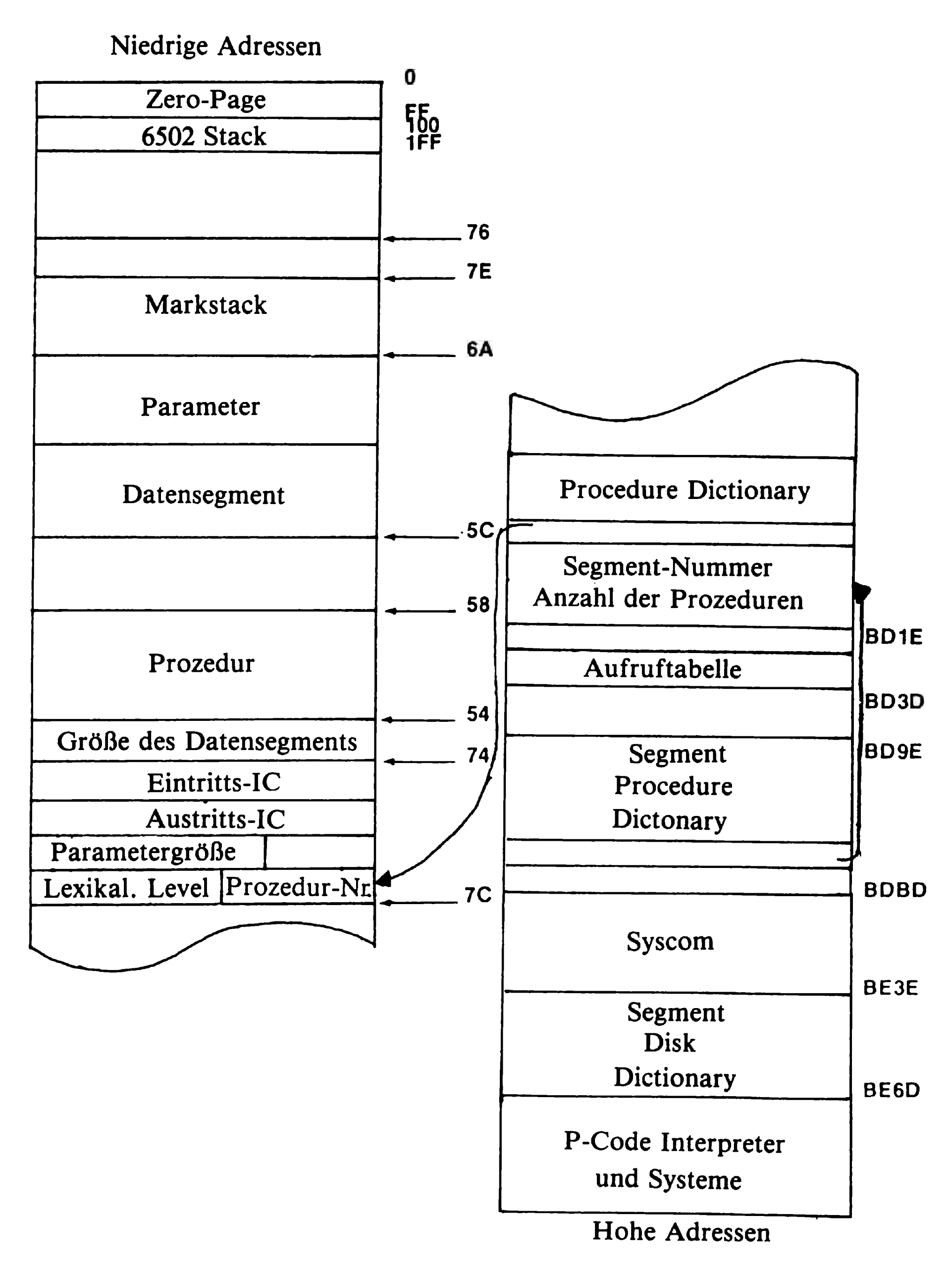


Abb. 1 Pascal Speicher-Adressen

Nützliche Pascal-Adressen

Die folgenden Adressen scheinen mit den Angaben in den Pascal-Handbüchern übereinzustimmen. Sie gelten sowohl für die alte als auch für die neue Version.

Adresse	Bedeutung
BDDE-BDFF	SYSCOM
BDDE,BD1E	IORESULT
BDE0,BDE1	XEQERR
BDE2	SYSUNIT
BDE3	BUGSTATE
BDE4,BDE5	GDIRP
BDEA,BDEB	STKBASE
BDEC,BDED	LASTMP
BDEE,BDEF	JTAB
BDF0,BDF1	SEG
BDF2,BDF3	BOMBP
BDF4,BDF5	BOMIPC
BDF6,BDF7	HLTLN
BDF8-BDFF	BRKPTS

Zero-Page-Zeiger

50,51	Zeiger auf BASE Activation Record
52,53	Markstack-Zeiger
54,55	JTAB-Zeiger
56,57	SEG-Zeiger
58,59	IPC
5A,5B	Pascal-Stack-Zeiger
5C,5D	Zeiger auf BASE-2
5E,5F	Wort-Offset vom Stack
64,65	Kopie von 50,51
6E-70	Sprungtabelle für P-Codes
71-73	Sprungtabelle für Sonderoperationen
74,75	Zwischenspeicher

78	Prozedurnummer zum Aufbau des Activation Records
7E,7F	MARKSTACK
80,81	Diskettenpuffer
86	Segmentnummer für Activation Record
8E,8F	Speicherplatz für Rücksprungadresse
90,91	Pascal-System
D0,DF	Zwischenspeicher für Diskzugriffe

Natürlich gibt es noch viel mehr Zero-Page-Adressen. Viel Spaß beim Suchen!

BE3E-BE9D Codefile-Puffer hat die Form: Laufwerk, Block-Nr., Länge

Dabei gibt "Laufwerk" die Unitnummer an, bei der das Segment zu finden ist. Die "Block-Nr." bezeichnet die Nummer des Diskettenblocks, bei der das Segment beginnt, und "Länge" gibt die Länge des Segments an. Es handelt sich dabei um ein Array mit 16 mal 3 Worten, das aufgebaut wird, nachdem der erste Block eines Codefiles gelesen wurde. Jede Wortgruppe bezieht sich dabei auf eines von maximal 16 möglichen Segmenten einer Datei.

BD1E-BD5D Aufrufzähler-Tabelle

Ein Aufrufzähler zeigt dem Betriebssystem an, wie oft ein bestimmtes Segment aufgerufen wurde. Für jedes Segment werden zwei Bytes reserviert, so daß man ein Segment über 65000 mal aufrufen kann, bevor Probleme auftreten.

BD5E-BD9D Weitere Datei

BD9E-BDBD Tabelle der Segment Procedure Dictionaries

Diese Tabelle zeigt dem Betriebssystem an, wo das zu einer Prozedur gehörige Segment Dictionary zu finden ist.

Die obengenannten Puffer werden von der Segment-Leseroutine (neue Version: E417; alte Version: E3D7) benutzt, die ein angegebenes Segment liest (0..F), das im Bereich BE3E-BE9D gespeichert wurde.

Die Seiten zwei und drei ($200 bis $3FF) werden von den Disketten-E/A-Prozeduren zur Fehlerüberprüfung verwendet. Dort abgelegte Werte werden bei Lese- oder Schreiboperationen auf der Diskette zerstört.

Philip Ender

PASCAL ZAP

Dieses Programm ist eine modifizierte Version des PASCAL ZAP-Programms, das in der Januarausgabe 1981 von *Call-A.P.P.L.E.* erschienen ist. Dieses Programm dient dazu, jeden beliebigen Block einer 16-Sektor-Diskette zu lesen, zu schreiben oder zu modifizieren. Bei Programmstart erscheint folgendes Befehlsmenü:

BEFEHLSUEBERSICHT

L(ESEN (BLOCK))
S(CHREIBEN (BLOCK))
P(UFFER ANZEIGEN)
D(RUCKEN DES PUFFERS)
A(SCII AENDERN)
H(EX AENDERN)
N(EUES LAUFWERK[4]
Q(UIT

IHRE WAHL:

Das Programm kann lesend oder schreibend auf jedes vom Benutzer angegebene Laufwerk zugreifen. Standardwert ist dabei Nr.4, das Boot-Laufwerk. Die Befehlsmöglichkeiten sind im wesentlichen selbsterklärend. Gibt man vom Menü aus ein "R" ein, so kann der Benutzer einen Diskettenblock von der durch die N(EUES LAUFWERK-Option vorbestimmten Diskette lesen. Der Block wird in einen Puffer kopiert, der hexadezimal und in ASCII-Schreibweise ausgegeben wird, wenn man "P" (wie P(UFFER ANZEIGEN) drückt. Mit H(EX oder A(SCII AENDERN können einzelne Bytes des Blocks geändert werden. Die Änderungen lassen sich mit S(CHREIBEN (BLOCK) auf die Diskette zurückschreiben.
Warnung: *Mit PASCAL ZAP sind irreparable Diskettenmodifikationen nicht auszuschließen.* Fertigen Sie von der Diskette, mit der Sie arbeiten, vor Gebrauch dieses Programms eine Sicherheitskopie an. PASCAL ZAP ist manchmal hilfreich beim Rekonstruieren

von Diskettendateien (das Directory beginnt bei Block 2). Hauptsächlich wird man das Programm dazu verwenden, einzelne Diskettenblocks zu untersuchen und damit mehr über die Arbeitsweise des Pascal-Betriebssystems zu lernen.

```
PROGRAM PASCAL_ZAP;

(**************************************************)
(*                                                *)
(* Von Philip B. Ender                            *)
(* Modifikationen von Ron DeGroat Juni '81        *)
(* Ersterscheinung in Call-A.P.P.L.E. ,1/81       *)
(*                                                *)
(**************************************************)

CONST SP=' ';

VAR
  BUF           : PACKED ARRAY[0..511] OF 0..255;
  HEX_DIGIT     : PACKED ARRAY[0..15 ] OF CHAR;
  HEX_BYTE      : PACKED ARRAY[0..1  ] OF CHAR;
  HEX_STR       : STRING[5];
  BLK_NUM, BYTE,
  DEV_NUM, DEC,
  NUM_COLS      : INTEGER;
  CHOICE,CH     : CHAR;
  PRINTER_OFF   : BOOLEAN;
  F             : INTERACTIVE;

PROCEDURE DEC_TO_HEX_BYTE(DEC: INTEGER);
BEGIN
  HEX_BYTE[0]:=HEX_DIGIT[(DEC DIV 16)];
  HEX_BYTE[1]:=HEX_DIGIT[(DEC MOD 16)]
END;

PROCEDURE WRITE_BLOCK;
BEGIN
  WRITELN; WRITELN;
  WRITE('"J" EINGEBEN, WENN BLOCK GESCHRIEBEN WERDEN SOLL ',
          BLK_NUM:3,SP:2);
  READLN(CH);
  IF CH='J' THEN
    UNITWRITE(DEV_NUM,BUF,512,BLK_NUM,0)
END;

PROCEDURE READ_BLOCK;
BEGIN
  WRITELN; WRITELN;
  WRITE('WELCHEN BLOCK LESEN?'); READLN(BLK_NUM);
  WRITELN;
  WRITE('LESE BLOCK ',BLK_NUM:3);
  UNITREAD(DEV_NUM,BUF,512,BLK_NUM,0)
END;
```

```
PROCEDURE DISPLAY_BUFFER;

VAR ROW,COL: INTEGER;

BEGIN
  ROW:=0; BYTE:=0;
  REPEAT
    WRITE(F,BYTE:3,':');
    FOR COL:=0 TO NUM_COLS DO
      BEGIN
        DEC_TO_HEX_BYTE(BUF[BYTE+COL]);
        WRITE(F,HEX_BYTE:3)
      END;
    WRITE(F,SP);
    FOR COL:=0 TO NUM_COLS DO
      IF (BUF[BYTE+COL]>31) AND
         (BUF[BYTE+COL]<127)
         THEN
             WRITE(F,CHR(BUF[BYTE+COL]))
         ELSE
           WRITE(F,'.');
    WRITELN(F);
    BYTE:=BYTE+NUM_COLS+1; ROW:=ROW+1;
    IF (ROW MOD 22 = 0) AND PRINTER_OFF THEN
        BEGIN
          WRITE(F,'BLOCK ',BLK_NUM:3,
                  ': SP-CONT; E-EXIT');
          READ(KEYBOARD,CH); PAGE(F);
          IF CH='E' THEN EXIT(DISPLAY_BUFFER)
        END
  UNTIL BYTE>504;
  WRITELN(F,'BLOCK ',BLK_NUM:3);
  IF NOT PRINTER_OFF THEN GOTOXY(7,7);
  WRITE('WEITER DURCH TASTENDRUCK');
  READ(KEYBOARD,CH);
END;


PROCEDURE PRINT_BUFFER;
BEGIN
  CLOSE(F); (* VOR RESET F SCHLIESSEN *)
  RESET(F,'PRINTER:');
  NUM_COLS:=15;
  PRINTER_OFF:=FALSE;
  WRITE('DRUCKEN....');
  DISPLAY_BUFFER;
  PRINTER_OFF:=TRUE;
  CLOSE(F);
  RESET(F,'CONSOLE:');
  NUM_COLS:=7;
END; (*PRINT_BUFFER*)


PROCEDURE GET_BYTE;
BEGIN
  WRITE('ZU AENDERNDES BYTE?');
  READLN(BYTE); WRITELN
END;
```

```
PROCEDURE ASCII_CHANGE;
BEGIN
 GET_BYTE;
 REPEAT
  IF (BUF[BYTE]>31) AND (BUF[BYTE]<127)
    THEN CH:=CHR(BUF[BYTE])
    ELSE CH:='.';
  WRITELN(BYTE:3,': ',CH,' = CHR(',BUF[BYTE],')');
  WRITE  (BYTE:3,': '); READ(CH);
  WRITELN;
  IF NOT EOLN THEN BUF[BYTE]:=ORD(CH);
  BYTE:=BYTE+1
 UNTIL EOLN
END;

PROCEDURE HEX_STR_TO_DEC;

VAR LO,HI: INTEGER;

BEGIN
  HI:=SCAN(16,=HEX_STR[1],HEX_DIGIT);
  LO:=SCAN(16,=HEX_STR[2],HEX_DIGIT);
  DEC:=HI*16+LO
END;

PROCEDURE HEX_CHANGE;

VAR LEN: INTEGER;

BEGIN
  GET_BYTE;
  REPEAT
    DEC_TO_HEX_BYTE(BUF[BYTE]);
    WRITELN(BYTE:3,': ',HEX_BYTE);
    WRITE  (BYTE:3,': ');
    READLN(HEX_STR);
    WRITELN;
    LEN:=LENGTH(HEX_STR);
    IF LEN<>0 THEN
      IF LEN>2 THEN
        WRITELN('HEX WERT ZU LANG')
      ELSE
        BEGIN
          IF LEN=1 THEN
          HEX_STR:=CONCAT('0',HEX_STR);
          HEX_STR_TO_DEC;
          BUF[BYTE]:=DEC; BYTE:=BYTE+1
        END
   UNTIL LEN=0
END;


PROCEDURE SET_DRIVE;
BEGIN
  GOTOXY(0,8);
  WRITE('BITTE UNIT (LAUFWERK) ANGEBEN (4..5, 9..12): ');
  READLN(DEV_NUM);
END;
```

```
PROCEDURE SHOWMENU;
BEGIN
  PAGE(OUTPUT);
  WRITELN; WRITELN;
  WRITELN(SP:5,'BEFEHLSUEBERSICHT');
  WRITELN;
  WRITELN(SP:5,'L(ESEN (BLOCK)');
  WRITELN(SP:5,'S(CHREIBEN (BLOCK)');
  WRITELN(SP:5,'P(UFFER ANZEIGEN');
  WRITELN(SP:5,'D(RUCKEN DES PUFFERS');
  WRITELN(SP:5,'A(SCII AENDERN');
  WRITELN(SP:5,'H(EX AENDERN');
  WRITELN(SP:5,'N(EUES LAUFWERK [',DEV_NUM,']');
  WRITELN(SP:5,'Q(UIT');
  WRITELN; WRITELN;
  WRITE('IHRE WAHL: ')
END;

PROCEDURE INITIALIZE;
BEGIN
  HEX_DIGIT:='0123456789ABCDEF';
  PRINTER_OFF:=TRUE;
  RESET(F,'CONSOLE:');
  NUM_COLS:=7;
  DEV_NUM:=4;
END;

BEGIN (*HAUPTPROGRAMM*)
  INITIALIZE;
  REPEAT
    SHOWMENU;
    READ(KEYBOARD,CHOICE);
    PAGE(OUTPUT);
    CASE CHOICE OF
      'L': READ_BLOCK;
      'S': WRITE_BLOCK;
      'P': DISPLAY_BUFFER;
      'D': PRINT_BUFFER;
      'A': ASCII_CHANGE;
      'H': HEX_CHANGE;
      'N': SET_DRIVE
    END;
  UNTIL CHOICE='Q';
  GOTOXY(8,8); WRITE('DAS WAR''S..');
END.
```

Ron DeGroat

Die narrensichere Befehlseingabe

Das häufigste und lästigste Problem für Benutzer von APPLE-Pascal ist wahrscheinlich das der Eingabefehler. In diesem Kapitel wird daher beschrieben, wie man die Eingabe von der Tastatur für jede Art von Programm praktisch narrensicher gestalten kann, indem man verhindert, daß unzulässige Daten ins Programm gelangen. Darüberhinaus wird bei Verwendung der hier wiedergegebenen Routinen (Listing Nr.1) die Dateneingabe interaktiver, indem jedes eingegebene Zeichen sofort auf seine Zulässigkeit hin überprüft wird.

Wartet ein APPLE-Pascal-Programm auf die Eingabe eines numerischen Wertes, so tritt bei versehentlicher Eingabe eines Buchstabens ein "BAD INPUT FORMAT"-Fehler auf. Der Computer fordert den Benutzer dann höflich auf "PRESS «SPACE» TO CONTINUE" (Bitte «Leertaste» drücken), worauf das System selber aber sehr unhöflich reagiert: es re-initialisiert sich selbst.

Offensichtlich nimmt das Betriebssystem Eingabe- oder Laufzeitfehler sehr ernst, denn es reagiert auf sie ziemlich dramatisch. Hinter diesem Verhalten steckt die Idee, daß Laufzeitfehler in Pascal-Programmen eigentlich gar nicht auftreten dürfen. In das Betriebssystem sind etliche Sicherheitsvorrichtungen eingebaut, die verhindern sollen, daß ein Programm "abstürzt". Normalerweise läuft ein Programm, das ohne Fehler kompiliert wurde, auch fehlerfrei. Eine wesentliche Ausnahme hiervon bildet die Klasse von Fehlern, die durch nicht erwartete Eingaben entstehen.

Unglücklicherweise erfordern die meisten Pascal-Programme die Eingabe von Daten durch einen menschlichen Benutzer mit Hilfe der Tastatur. Da nun aber Menschen dazu neigen, Fehler zu machen, und Pascal auf solche Fehler heftig reagiert, ist es äußerst wichtig, über Tastatureingabe-Prozeduren zu verfügen, die die Eingabe unzulässiger Daten so zuverlässig wie nur irgend möglich verhindern. Die einfachste Lösung dieses Problems ist, die eingegebene Zeile nach Drücken der Return-Taste auf Zulässigkeit hin zu prüfen. Der interaktivere Weg besteht darin, jedes einzelne Zeichen

sofort nach der Eingabe auf seine Legalität hin zu testen.

Die Tastatureingabe kann darüberhinaus verbessert werden, indem man mit Hilfe der Backspace-Taste (Linkspfeil) jeden einzelnen Wert korrigiert, bevor man ihn durch Drücken von Return dem Programm übergibt. Im APPLE-Pascal wird dieser Mechanismus bei der Eingabe von Strings unterstützt, nicht aber bei Real- und Integer-Zahlen. Zur Lösung dieses Problems kann man die Daten als Strings einlesen und dann in die entsprechenden numerischen Werte konvertieren. Zusammenfassend kann man sagen, daß eine gute Eingaberoutine unzulässige Eingaben zur Vermeidung von Laufzeitfehlern herausfiltert, die einzelnen gedrückten Tasten jeweils auf Legalität hin überprüft, damit das Programmverhalten interaktiver wird, und daß die Zahleneingabe in Form von Strings erfolgt, so daß die Backspace-Taste zu Korrekturzwecken benutzt werden kann.

Das Einlesen von Fließkommazahlen (Real-Zahlen) ist am schwierigsten zu realisieren (vgl. PROCEDURE GETFPNUM), da das Eingabeformat bei diesem Datentyp so unterschiedlich sein kann. Das erste Zeichen kann zum Beispiel in der Menge [''0''..''9'',''-'',''.''] sein, danach ist jedoch die Eingabe eines Minuszeichens nicht mehr zulässig. Eine Real-Zahl darf auch nur einen Dezimalpunkt enthalten und eine gewisse Länge nicht überschreiten. Darüberhinaus darf eine solche Zahl nur in einem bestimmten, durch Minimal- und Maximalwerte begrenzten Bereich liegen. Beachtet man all diese Beschränkungen, so ist es relativ einfach, die meisten durch Eingabefehler verursachten Laufzeitfehler zu vermeiden.

Wenn man die in Listing Nr.1 gezeigte intrinsische Unit kompiliert und in die SYSTEM.LIBRARY eingebaut hat, kann man sie bequem von jedem beliebigen Pascal-Programm aufrufen, indem man die Deklaration ''USES KYBDSTUFF'' angibt (also in der gleichen Weise, wie man auch auf die TURTLEGRAPHICS- oder APPLESTUFF-Unit zugreifen kann). Intrinsische Units sind nicht nur einfach zu benutzen, sondern sparen auch Diskettenplatz, da sie nur in der SYSTEM.LIBRARY existieren. Manchmal ist jedoch von der Verwendung solcher Units abzuraten, nämlich dann, wenn ein Programm von der SYSTEM.LIBRARY unabhängig sein soll. In diesem Fall sollte man reguläre Units verwenden. Kommt es wegen der Programmlänge zu Speicherplatzproblemen, so sollte man nur diejenigen Prozeduren direkt in den Programmtext kopieren, die man auch tatsächlich braucht.

Das KYBDDEMO-Programm, das in Listing Nr.2 abgedruckt ist, zeigt, wie man KYBDSTUFF verwenden kann. Mit KYBDDEMO kann man auch testen, wie effizient die Eingaberoutinen sind. Der schnellste Weg, herauszufinden, ob die Prozeduren auch wirklich absolut narrensicher sind, ist, sie von anderen Benutzern ausprobieren zu lassen. Vorschläge zur Verbesserung und Erweiterung dieser Prozeduren werden selbstverständlich gerne angenommen.

Durch die in Listing Nr.1 abgedruckten Prozeduren wird die Tastatureingabe sehr sicher und interaktiv. Mit ihnen kann man benutzerfreundliche Programme schreiben, die so gut wie immun gegen Eingabefehler sind.

GETCHAR ist die allgemeinste und nützlichste dieser Routinen. Sie prüft jedes Zeichen bei seiner Eingabe auf Zulässigkeit. Dabei werden nur die in OKSET befindlichen Zeichen akzeptiert. Werden andere, nicht im Set aufgenommene Zeichen eingegeben, erfolgt eine direkte Warnung für den Benutzer mittels eines Lautsprechertons. Da jedes Zeichen einzeln geprüft wird, ist die Benutzereingabe völlig interaktiv. GETCHAR wird von den anderen Routinen benutzt, um Strings, Hexadezimal-, Integer- und Realzahlen zu lesen.

Zur Zeit gibt es die größeren Schwierigkeiten, die EDV zu verbessern, im Software- und nicht im Hardware-Bereich. Vielleicht könnte man aus den Bereichen der Hardware-Entwicklung etwas für die Software-Entwicklung lernen.

In der Elektronik hat der integrierte Schaltkreis die Technik durch die Modularisierung von Grundschaltungen revolutioniert. Durch das IC, das einen Grundbaustein der Elektrotechnik darstellt, ließen sich komplizierte elektronische Geräte billiger und einfacher herstellen. Sogar einfache Schaltkreise wurden durch die effizienteren, einsteckbaren Chips verbessert.

Im Computerbereich ist eine ähnliche Revolution auf dem Software-Sektor nötig. Durch effiziente, gut konstruierte Software-Blöcke könnte man nämlich die Entwicklung von Programmen enorm vorantreiben. Mit hochwertigen, einsteckbaren Modulen könnte man eine Menge doppelter Arbeit sparen und eine ganze Reihe mittelmäßiger Routinen durch gut getestete, standardisierte Prozeduren ersetzen. Mit solchen Programmblöcken ausgerüstet kann dann auch ein zweitklassiger Programmierer erstklassige Programme schreiben.

Es bleibt zu hoffen, daß die hier vorgestellten Routinen in APPLE-Kreisen einen Maßstab setzen, was die Entwicklung standardisierter Software-Tools mit universellen Anwendungsmöglichkeiten betrifft. Mit guten Werkzeugen erleichtert man sich die Arbeit und erhält bessere Resultate.

```
(************************)
(* Listing #2: KYBD DEMO *)
(*                      *)
(* von Ron De Groat     *)
(************************)

(*$V-*)
PROGRAM KYBDDEMO;

USES KYBDSTUFF;


VAR ALPHABET : SETOFCHAR; (*Typdefinition in KYBDSTUFF*)
    FLOAT_PT : REAL;
    INT_NUM,
    HEX_VAL    : INTEGER;
    HEXWORD    : STRING[2];
    ALFASTRING : STRING[10];
    CH         : CHAR;

BEGIN

  REPEAT
    WRITELN('*** WEITER DURCH DRUECKEN EINER TASTE ***');
  UNTIL KEYPRESS;
  READ(KEYBOARD,CH); (*Zum Loeschen des Tastendrucks*)

  PAGE(OUTPUT);
  PROMPTAT(0,'VERSUCHEN SIE, UNZULAESSIGE WERTE EINZUGEBEN.');

  ALPHABET:=['A'..'Z'];
  PROMPTAT(4,'GEBEN SIE EIN MAXIMAL DREISTELLIGES BUCHSTABENSTRING EIN: ');
  GETSTRING(ALFASTRING,ALPHABET,3);

  WRITELN;
  INT_NUM:=GETINTEGER('INTEGER (-1<I<100): ',100,-1);

  WRITELN;
  FLOAT_PT:=GETFPNUM('FLIESSKOMMAZAHL (0.1<Z<9.999): ',9.999,0.1);

  WRITELN;
  HEX_VAL:=GET_HEX_VAL('HEX ADDR: ',4);
  INTTOHEX(HEXVAL,HEXWORD);
  WRITELN('HEX WERTE WERDEN ALS INTEGERZAHLEN ZURUECKGEGEBEN: ',HEXVAL);
  WRITELN('UND KOENNEN WIEDER IN HEX UMGEWANDELT WERDEN:  ',HEXWORD);

  WRITELN;
  HEX_VAL:=GETHEXVAL('HEX BYTE:  ',2);
  WRITELN('HEX WERT: ',HEX_VAL);

END.
```

```
(***************************************)
(*                                     *)
(* Listing #1: KYBDSTUFF               *)
(*                                     *)
(* Von Ron DeGroat   Juli 1981         *)
(*                                     *)
(***************************************)

(*$S+,V-*)
UNIT KYBDSTUFF; INTRINSIC CODE 25 DATA 26;

(* Durch Loeschen von "INTRINSIC...26;" wird KYBDSTUFF zur
   regulaeren Unit. Solche Units koennen mit dem Linker
   explizit in ein Hauptprogramm eingebunden werden. In-
   trinsische Units muessen in die SYSTEM.LIBRARY eingefuegt
   werden. *)


INTERFACE

TYPE SETOFCHAR = SET OF CHAR; (* Muss fuer Parameterliste deklariert werden *)

(* Die folgenden Befehle koennen vom Hauptprogramm in WRITE-Anweisungen      *)
(* verwendet werden. Die Variablennamen duerfen nicht im Hauptprogramm       *)
(* erneut deklariert werden. *)

VAR CR,         (*Carriage return (Wagenruecklauf)  *)
    BS,         (*Back space (Rueckwaertstaste)     *)
    EOL,        (*Loeschen bis Zeilenende           *)
    EOS,        (*Loeschen bis Bildschirmende       *)
    BELL:CHAR;  (*Piepton                           *)
FUNCTION  GET_CHAR(OKSET:SETOFCHAR):CHAR;
PROCEDURE GETSTRING(VAR S:STRING; OKSET:SETOFCHAR; MAXLEN:INTEGER);
FUNCTION  GET_HEX_VAL(PROMPT:STRING; MAXLEN:INTEGER):INTEGER;
FUNCTION  HEX_TO_INT(HEXSTR:STRING):INTEGER;
PROCEDURE INT_TO_HEX(INT:INTEGER; VAR HEX_STR:STRING);
FUNCTION  FP_NUM(FP_STR:STRING):REAL;
FUNCTION  GET_FP_NUM(PROMPT:STRING; MAXVAL,MINVAL:REAL):REAL;
FUNCTION  GET_INTEGER(PROMPT:STRING; MAXVAL,MINVAL:INTEGER):INTEGER;
PROCEDURE PROMPTAT(LINE:INTEGER; MESSAGE:STRING);
FUNCTION  KEYPRESS:BOOLEAN;


IMPLEMENTATION

VAR HEXDIGIT     :PACKED ARRAY[0..15] OF CHAR;


(**************************************************)
(* LIEST OKSET-ZEICHEN, BEI ALLEN ANDEREN PIEPT'S *)
(**************************************************)

FUNCTION GET_CHAR(*(OKSET:SETOFCHAR):CHAR*);

VAR CH    :CHAR;
    GOOD  :BOOLEAN;
```

```
BEGIN
  REPEAT
    READ(KEYBOARD,CH);
    IF EOLN(KEYBOARD) THEN CH:=CR;
    GOOD:=CH IN OKSET;
    IF NOT GOOD THEN WRITE(BELL)
      ELSE IF CH IN [' '..CHR(125)]
             THEN WRITE(CH);
  UNTIL GOOD;
  GET_CHAR:=CH;
END;


(***********************************************************)
(* LIEST STRING MIT LAENGE "MAXIMUM" UND ZEICHEN AUS "OKSET" *)
(***********************************************************)

PROCEDURE GETSTRING(*VAR S:STRING; OKSET:SETOFCHAR; MAXLEN:INTEGER*);

VAR S1         :STRING[1];
    STEMP      :STRING;
    LEN        :INTEGER;
    FIRSTCHAR  :BOOLEAN;
    LASTCHAR   :BOOLEAN;
    GETSET     :SETOFCHAR;

BEGIN
 S1:=' '; STEMP:='';
 IF MAXLEN<1 THEN MAXLEN:=1
 ELSE IF MAXLEN>255 THEN MAXLEN:=255;
 REPEAT
  LEN:=LENGTH(STEMP);
  FIRSTCHAR:=(LEN=0);
  LASTCHAR:=(LEN=MAXLEN);

  IF FIRSTCHAR THEN GETSET:=OKSET
  ELSE IF LASTCHAR THEN GETSET:=[CR,BS]
       ELSE GETSET:=OKSET+[CR,BS];

  S1[1]:=GETCHAR(GETSET);

  IF S1[1] IN OKSET THEN STEMP:=CONCAT(STEMP,S1)
  ELSE IF S1[1]=BS THEN
    BEGIN
     WRITE(BS,' ',BS);
     DELETE(STEMP,LEN,1);
    END;
  UNTIL S1[1]=CR; WRITELN;
 S:=STEMP;
END;


(**********************************************)
(* GIBT NACHRICHT "MESSAGE" IN ZEILE "LINE" AUS *)
(**********************************************)
```

```
PROCEDURE PROMPTAT(*LINE:INTEGER; MESSAGE:STRING*);
BEGIN
 GOTOXY(0,LINE);
 WRITE(MESSAGE,EOL);
END;

(*****************************************)
(* KONVERTIERT INTEGERZAHL IN HEX STRING *)
(*****************************************)

PROCEDURE INT_TO_HEX(*(INT:INTEGER; VAR HEX_STR:STRING)*);

VAR HIBYTE,LOBYTE :INTEGER;

BEGIN
 HEX_STR:='0000';

 IF INT<0 THEN
   BEGIN
     INT:= INT+32767+1;;
     HIBYTE:= (INT DIV 256) +128;
   END
 ELSE
   HIBYTE:= INT DIV 256;
   LOBYTE:= INT MOD 256;

 HEXSTR[1]:= HEXDIGIT[HIBYTE DIV 16];
 HEXSTR[2]:= HEXDIGIT[HIBYTE MOD 16];
 HEXSTR[3]:= HEXDIGIT[LOBYTE DIV 16];
 HEXSTR[4]:= HEXDIGIT[LOBYTE MOD 16];

 WHILE (HEXSTR[1]='0') AND (LENGTH(HEXSTR)>1) DO DELETE(HEXSTR,1,1);

END;


(*************************************)
(* CONVERTIERT HEX STRING IN INTEGER *)
(*************************************)

FUNCTION HEX_TO_INT(*(HEXSTR:STRING):INTEGER*);

VAR I,NUM,DIGIT  :INTEGER;

BEGIN
 NUM:=0;
 FOR I:=1 TO LENGTH(HEXSTR) DO
 BEGIN
   DIGIT:=SCAN(16,=HEXSTR[I],HEXDIGIT);
   NUM:=NUM*16+DIGIT;
  END;
 HEX_TO_INT:=NUM;
END;

(******************************************)
(* LIEST HEX-ZAHL UND LIEFERT INTEGERWERT  *)
(******************************************)

FUNCTION GET_HEX_VAL(*(PROMPT:STRING; MAXLEN:INTEGER):INTEGER*);
```

```
VAR HEXSET  :SETOFCHAR;
    HEX_STR :STRING;

BEGIN
  WRITE(PROMPT,EOL);
  HEXSET:=['0'..'9','A'..'F'];
  IF MAXLEN>4 THEN MAXLEN:=4;
  GETSTRING(HEX_STR,HEXSET,MAXLEN);
  GET_HEX_VAL:=HEX_TO_INT(HEX_STR);
END;


(*********************************************)
(* CONVERT FLIESSKOMMA-STRING IN REALWERT    *)
(*********************************************)

FUNCTION FP_NUM(*(FP_STR:STRING):REAL*);

VAR POWER,SIGN,I     :INTEGER;
    NUM              :REAL;

BEGIN
  IF FP_STR[1]='-' THEN
    BEGIN
      SIGN:=-1;
      DELETE(FP_STR,1,1);
    END
  ELSE SIGN:=1;

  POWER:=POS('.',FP_STR);
  IF POWER<>0 THEN
    BEGIN
      DELETE(FP_STR,POWER,1);
      POWER:=LENGTH(FP_STR)-POWER+1;
    END;

  NUM:=0;
  FOR I:=1 TO LENGTH(FP_STR) DO
    NUM:=10*NUM+(ORD(FP_STR[I])-ORD('0'));
  FP_NUM:=SIGN*NUM/PWROFTEN(POWER);

END;

(**********************************************)
(* WIRD VON GET_FP_NUM AND GET_INTEGER BENUTZT *)
(**********************************************)

FUNCTION GET_NUM(MAXVAL,MINVAL:REAL; PT_OK:BOOLEAN):REAL;

VAR FIRSTCHAR, LASTCHAR   :BOOLEAN;
    NUMSET, GETSET, OKSET :SETOFCHAR;
    NUM_STR_TEMP          :STRING[10];
    S1                    :STRING[1];
    LEN, MAXLEN           :INTEGER;
    NUM                   :REAL;

BEGIN
  NUMSET:=['0'..'9'];
  IF MINVAL<0 THEN OKSET:=NUMSET+['-'] ELSE OKSET:=NUMSET;
```

```
  IF PT_OK THEN MAXLEN:=8 ELSE MAXLEN:=7;
  S1:=' '; NUM_STR_TEMP:='';

  REPEAT
    LEN:=LENGTH(NUM_STR_TEMP);
    FIRSTCHAR:=(LEN=0);
    LASTCHAR:=(LEN=MAXLEN);

    IF PT_OK THEN OKSET:=OKSET+['.']
    ELSE OKSET:=OKSET-['.'];

    IF FIRSTCHAR THEN GETSET:=OKSET
    ELSE
      IF LASTCHAR THEN GETSET:=[CR,BS]
      ELSE GETSET:=OKSET+[CR,BS]-['-'];

    S1[1]:=GET_CHAR(GETSET);
    IF S1='.' THEN PT_OK:=FALSE;

    IF S1[1] IN OKSET THEN
      BEGIN
        NUM_STR_TEMP:=CONCAT(NUM_STR_TEMP,S1);
        IF S1[1] IN NUMSET THEN
          BEGIN
            NUM:=FP_NUM(NUM_STR_TEMP);
            IF (NUM>MAXVAL) OR (NUM<MINVAL) THEN
              BEGIN
                WRITE(CHR(7));
                WRITE(BS,' ',BS);
                DELETE(NUM_STR_TEMP,LEN+1,1);
              END;
          END;
      END

    ELSE IF S1[1]=BS THEN
      BEGIN
        IF POS('.',NUM_STR_TEMP)=LEN THEN
          PT_OK:=TRUE;
        WRITE(BS,' ',BS);
        DELETE(NUM_STR_TEMP,LEN,1);
      END;

   UNTIL S1[1]=CR; WRITELN;

   GET_NUM:=FP_NUM(NUM_STR_TEMP);

END; (*GET_NUM*)


(*****************************************************)
(* LIEST FLIESSKOMMAZAHL IM BEREICH VON "MIN" BIS "MAX" *)
(*****************************************************)

FUNCTION GET_FP_NUM(*(PROMPT:STRING; MAXVAL,MINVAL:REAL):REAL*);

VAR POINT_OK:BOOLEAN;
```

```
BEGIN
  WRITE(PROMPT,EOL);
  POINT_OK:=TRUE;
  GET_FP_NUM:=GET_NUM(MAXVAL,MINVAL,TRUE);
END;

(*********************************************)
(* LIEST INTEGERZAHL IM BEREICH "MIN" BIS "MAX" *)
(*********************************************)

FUNCTION GET_INTEGER(*(PROMPT:STRING; MAXVAL,MINVAL:INTEGER):INTEGER*);

VAR POINT_OK:BOOLEAN;

BEGIN
  WRITE(PROMPT,EOL);
  POINT_OK:=FALSE;
  GET_INTEGER:=TRUNC(GET_NUM(MAXVAL,MINVAL,FALSE));
END;

(*************************************)
(* PASCAL VERSION DER KEYPRESS-FUNKTION *)
(* FUNKTIONIERT GGF. NICHT MIT EXTERNER *)
(* KONSOLE BZW. 80 ZEICHEN-KARTE       *)
(*************************************)

FUNCTION KEYPRESS(*:BOOLEAN*);

TYPE BYTE=0..255;
     PA=PACKED ARRAY[0..1] OF BYTE;

VAR MEMREF:RECORD CASE BOOLEAN OF
             TRUE: (ADDR:INTEGER);
             FALSE:(BYTE:^PA);
           END;
    RPTR,WPTR,KEYBD:BYTE;

BEGIN
    MEMREF.ADDR:=-16384;  (*KEYBOARD INPUT PORT*)
    KEYBD:=MEMREF.BYTE^[0];

    MEMREF.ADDR:=-16616;  (*PUFFERZAEHLER*)
    RPTR:=MEMREF.BYTE^[0];  (*INPUT *)
    WPTR:=MEMREF.BYTE^[1];  (*OUTPUT*)

    KEYPRESS:=(KEYBD > 127) OR (RPTR <> WPTR);

END;

BEGIN        (*HAUPTPROGRAMM*)

  CR:=CHR(13);    (*ctl-M*)
  BS:=CHR(8);     (*ctl-H*)
  EOL:=CHR(29);   (*ctl-]*)
  EOS:=CHR(11);   (*ctl-K*)
  BELL:=CHR(7);
  HEXDIGIT:='0123456789ABCDEF';

END.
```

Allen W. Todd

Aufbau eines Pascal-Disk-Directories

Haben Sie schon einmal versucht, von einem Programm aus auf das Dateiverzeichnis einer Pascal-Diskette (Directory) zuzugreifen? Das ist eigentlich ganz einfach, wenn man zwei Dinge weiß: a) wo sich das Directory befindet, und b) wie das Directory aufgebaut ist. Die Antwort auf die erste Frage läßt sich ziemlich leicht herleiten. Wenn man sich mit dem Filer ein E(rweitertes Listing einer Diskette ausgeben läßt, stellt man fest, daß die erste Datei bei Block 6 beginnt. Damit bleiben die Blöcke 0 bis 5 als wahrscheinlicher Platz für das Directory übrig. Tatsächlich liegt das Dateiverzeichnis in den Blöcken 2 bis 5; die Blöcke 0 und 1 werden für den zweiten Teil des Bootvorganges benötigt. Die eigentliche Directory-Struktur läßt sich nicht ganz so leicht erkennen. Hat man diese jedoch erst einmal herausgefunden, besticht sie durch ihren eleganten, gleichzeitig jedoch einfachen, Aufbau.

Die Struktur eines UCSD-Pascal-Directories wird in den Deklarationen von Tabelle 1 dargestellt. Danach besteht ein Pascal-Directory aus 0 bis 77 varianten Records des Typs "direntry". Directory-Einträge werden sequenziell gespeichert und beginnen auf jeder Diskette in Block 2. Jeder Eintrag besteht aus einem varianten Record, der zwei mögliche Varianten hat. Die aktuelle Variante wird durch ihr Tag-Feld bestimmt. Bei "direntry" ist dieses Tag-Feld dfkind (filetype). Ist der Dateityp "volume" oder "secure", handelt es sich bei dem Directory-Eintrag um Informationen, die die gesamte Diskette betreffen. Dieser Eintrag ist der erste Directory-Eintrag (direntry [0]) und enthält die in Tabelle 2 aufgelisteten Informationen.

Wie man diese Informationen benutzen kann, zeigt Ihnen die FUNCTION "getdirectory" in dem nachstehenden Programmlisting. Diese Funktion versucht, das Directory von Laufwerk "dunit" zu lesen. Wenn "getdirectory" ein Pascal-Dateiverzeichnis auf "dunit" findet, liefert es einen Zeiger auf das Directory und aktualisiert die Stringvariable "vname", die danach den Namen der gefundenen Diskette enthält. Wird kein Directory gefunden oder kann das Directory nicht gelesen werden, so wird dem Zeiger der Wert "nil" zugewiesen und "vname" nicht aktualisiert. Wenn man fest-

stellt, daß ein Directory gelesen wurde, kann man den Zeiger benutzen, um auf die gewünschten Directory-Informationen zuzugreifen.

Tab. 1

```
TYPE
 vid           = string [7]        (Diskettenname)
 fid           = string [15]       (Dateiname)
 daterec       = PACKED RE-
               CORD                (Datum-Record, 16 Bits)
                mo : 1..12;
                day : 1..31;
                year: 0..99;
               END;
 filekind      = (vol,             (Eintrag des Diskettenna-
                                   mens)
                badfile,           (Datei mit defekten Disket-
                                   tenblöcken)
                code,              (Codedatei, vom Rechner
                                   ausführbar)
                text,              (Textdatei, normal lesbar)
                info,              (Informationsdatei, für De-
                                   bugger)
                data,              (Datendatei)
                graf,              (Grafische Vektoren)
                foto,              (HI-RES-Bild)
                secure);           (Nicht benutzt)

 direntry      = RECORD
dfirstblock    : INTEGER;          (Erster Dateiblock)
dlastblock     : INTEGER;          (Letzter Dateiblock)
CASE dfkind    : filekind OF       (Dateityp)
  vol, secure :                    (Variante für Laufwerksein-
                                   trag)
  (dvid        : vid;              (Diskettenname)
  deovblk      : INTEGER;          (Anzahl der Blöcke)
  dfilenum     : INTEGER;          (Anzahl der Dateien)
  ddummy       : INTEGER;          (Dummy)
  dlastboot    :daterec);          (Datum des letzten Bootvor-
                                   gangs)
```

```
badfile, code,
text, info, data,
graf, foto :                                     (Variante für Dateieinträge)
  (dfid          : fid;                          (Dateiname)
  dlastbyte       1..512;                        (EOF-Byte)
  daccess        : daterec);                     (Datum des letzten Zugriffs)
END;             (direntry)
directory        = ARRAY [0..77] O
                 direntry; (max. 77 Da-
                 teien pro Disk)
```

Tab. 2

Feld	**Datentyp**	**Typischer Wert**
Erster Block des Disketten-eintrags	Integer	0
Letzter Block des Disket-teneintrags	Integer	6
Eintragstyp (filekind)	Integer	0 oder 8
Diskettenname	String [7]	"APPLE1"
Anzahl der Blöcke auf der Diskette	Integer	280
Anzahl der Dateien auf der Diskette	Integer	0..76
Dummy (nur zum Platzauf-füllen)	Integer	0
Datum des letzten Bootens	Daterec	(beliebiges Datum)

Die übrigen Directory-Einträge enthalten Informationen über die einzelnen auf der Disk gespeicherten Dateien. Siehe Tabelle 3.

Tab. 3

Feld	**Datentyp**	**Typischer Wert**
Erster Block der Datei	Integer	6..279
Letzter Block der Datei	Integer	6..279
Dateityp (filekind)	Integer	1..7
Dateiname	String [15]	"SYSTEM.APPLE"
Anzahl der Bytes im letzten Block	Integer	512
Datum der letzten Änderung	Daterec	(beliebiges Datum)

Daten	**Start-Byte**	**End-Byte**	**Länge**	**Format**	**Bemerkungen**
Marker	4	83	80	Packed Char.	10 mal, jedes Element mit 8 Bytes
Marker-Adressen	94	113	20	Integer	10 mal, jedes Element mit 2 Bytes
Auto indent	114	115	2	Boolean	Nur Hi-Bit wird benutzt
Filling	116	117	2	Boolean	Nur Hi-Bit wird benutzt
Token def.	118	119	2	Boolean	Nur Hi-Bit wird benutzt
Linker Rand	120	121	2	Integer	
Rechter Rand	122	123	2	Integer	
Absatzrand	124	125	2	Integer	
Befehlszeichen	126	127	2	Integer	
Anfangsdatum	128	129	2	Packed Rec.	
Monat					erste vier Bits
Tag					nächste fünf Bits
Jahr					letzte sieben Bits
Letzter Zugriff	130	131	2	Packed Rec.	
Monat					erste vier Bits
Tag					nächste fünf Bits
Jahr					letzte sieben Bits

```
PROGRAM DIR;
TYPE
   pointer = ^directory;
   vid = string[7];
   fid = string[15];
   daterec = PACKED RECORD
               mo:    1..12;
               day:   1..31;
               year:  0..99;
             END;
   filekind = (vol, badfile, code, text, info, data, graf, foto, secure);
   direntry = RECORD
                dfirstblock:  INTEGER;
                dlastblock:   INTEGER;
                CASE dfkind:  filekind OF
                   vol, secure:  (dvid:        vid;
                                  deovblk:     INTEGER;
                                  dfilenum:    INTEGER;
                                  ddummy:      INTEGER;
                                  dlastboot:   daterec);
                   badfile, code, text, info, data, graf, foto:
                                 (dfid:        fid;
                                  dlastbyte:   1..512;
                                  daccess:     daterec)
                END;
   directory = ARRAY[0..77] OF direntry;

VAR
   dirptr: ^directory;
   I, device:  INTEGER;
   volname:  string;

FUNCTION getdirectory ( dunit: INTEGER; VAR vname: string):pointer;

BEGIN
   getdirectory:=NIL;
   new(dirptr);
   (*$I-*)
   unitread (dunit,dirptr^,sizeof(directory),2);
   (*$I+*)
   IF ioresult = 0
      THEN WITH dirptr^[0] DO
         BEGIN
            IF  (dfkind IN [vol,secure])
                AND (dfirstblock=0)
                AND (dlastblock=6)
                THEN
                   BEGIN
                      vname:=dvid;
                      getdirectory:=dirptr;
                   END;
         END;  (*with*)
  END;  (*getdirectory*)
```

```
BEGIN  (*Hauptprogramm*)
   write('Directory von welchem Laufwerk lesen: ');
   readln(device);
   IF getdirectory(device,volname)=NIL
   THEN writeln('Directory nicht gefunden')
   ELSE BEGIN
          writeln(volname,':');
          FOR I:= 1 TO dirptr^[0].dfilenum DO
            WITH dirptr^[I], daccess DO
              BEGIN
                WHILE length(dfid)<15 DO dfid:=concat(dfid,' ');
                writeln(dfid,dlastblock-dfirstblock:4,
                        mo:4,'/',day:2,'/',year:2,dfirstblock:5);
              END;
          END;
END.
```

Alan J. Nayer

Der Textdatei-Kopf

Viele Benutzer des APPLE-Pascal-Betriebssystems und der Sprache Pascal sind mit dem Format vertraut, in dem Textdateien gespeichert werden. Besonders diejenigen unter Ihnen, die Programme geschrieben oder den Editor zur Textverarbeitung benutzt haben, werden mit dem grundsätzlichen Aufbau von Textdateien vertraut sein.

Dieses Kapitel setzt ein allgemeines Wissen über das Textdatei-Format und Kenntnisse über die "Environment"-Parameter des Pascal-Editors voraus. Zur Auffrischung Ihrer Kenntnisse schlage ich vor, sich den kurzen Abschnitt über Textdateien im "APPLE Pascal Operating System Reference Manual" (S.266 in der aktuellen Ausgabe) durchzulesen. Das Handbuch beschreibt allerdings nicht, wie die Textdatei-Kopfseite aufgebaut ist; ich möchte daher im folgenden etwas näher darauf eingehen.

Textdateien bestehen aus sog. Seiten, wobei eine Seite zwei aneinandergrenzende Diskettenblocks enthält; da ein Block 512 Bytes umfaßt, entspricht dies 1024 Bytes. Textdateien bestehen immer aus einer ganzen Anzahl von Seiten, sie setzen sich daher auch immer aus einer geraden Block-Anzahl zusammen. Die erste Seite ist der Textkopf, der fast vollständig leer ist und vom Betriebssystem mit binären Nullen gefüllt wird: Tatsächlich enthält nur rund ein Achtel des Textkopfes sinnvolle Informationen.

Diese Daten machen das "Environment", die Umgebung der Textdatei, aus. Man kann sie sich mit dem S(et-Befehl des Editors ansehen und auch verändern. Eine Liste dieser Daten finden Sie in Abb.1.

Die ersten signifikanten Daten beginnen bei Byte 4. (Beachten Sie bitte: alle Offsets werden von Byte Nr.0 an gezählt, Byte 4 ist damit tatsächlich das fünfte Byte der Kopfseite.) An dieser Stelle werden die zehn Marker gespeichert, die von 0 bis 9 durchnummeriert sind und dazu dienen, bestimmte Stellen der Textdatei zu markieren. Sie werden hintereinander gespeichert; der Name von Marker 0 reicht von den Bytes 4 bis 11, der Name von Marker 1 von 12 bis 19 usw. Die Marker werden vom Editor aus eingegeben und auf acht Stellen gestutzt oder mit Leerzeichen aufgefüllt, so daß der Name achtstellig wird.

Der Editor verbindet jeden Marker-Namen mit einer bestimmten Byte-Adresse, auf die der Cursor gesetzt wird, wenn man den entsprechenden

Marker eingibt. Wenn beispielsweise der Cursor beim Setzen eines Markers am Anfang der Textdatei steht, ist eins die zugehörige Marker-Adresse. Die Marker-Adressen werden als 2-Byte Integer-Zahlen beginnend bei Byte 94 des Textkopfes gespeichert. Erwartungsgemäß bezieht sich die erste Marker-Adresse auf den ersten Marker-Namen (Bytes 4 bis 11) usw.

Unmittelbar auf die Marker-Adressen folgen drei Worte mit Boole'schen Daten. Von diesen drei Worten ist jedoch nur jeweils das höchstwertige Bit signifikant. Mit diesen drei Boole'schen Werten werden die Zustände der Optionen "Auto Indent", "Filling" und "Token" dargestellt.

Die nächsten drei Felder enthalten Integer-Zahlen, die die Textgrenzen bestimmen: linker Rand, rechter Rand und Absatzrand. Das Befehlszeichen wird in den nächsten beiden Bytes im ASCII-Format gespeichert.

Die letzten beiden Einträge sind Kalenderdaten: zuerst das Datum der Textdatei-Entstehung, danach das Datum der letzten Änderung. Beide Daten werden vom Betriebssystem anhand des aktuellen Diskettendatums festgelegt und sind in einem extrem verdichteten Format abgelegt, das im ganzen Betriebssystem Verwendung findet. Der Monat, Tag und das Jahr werden in den ersten vier, nächsten fünf und letzten sieben Bits eines Wortes gespeichert.

Listing Nr.1 enthält ein Programm mit dem Namen TEXTINFO, das die Textkopfseite einer beliebigen Textdatei abfragt und die dort enthaltenen Daten auf dem Bildschirm und/oder einem Drucker ausgibt.

Daten	Start-Byte	End-Byte	Länge	Format	Bemerkungen
Marker	4	83	80	Packed Char	10 mal, jedes Element mit 8 Bytes
Marker-Adressen	94	113	20	Integer	10 mal, jedes Element mit 2 Bytes
Auto indent	114	115	2	Boolean	Nur Hi-Bit wird benutzt
Filling	116	117	2	Boolean	Nur Hi-Bit wird benutzt
Token def.	118	119	2	Boolean	Nur Hi-Bit wird benutzt
Linker Rand	120	121	2	Integer	
Rechter Rand	122	123	2	Integer	
Absatzrand	124	125	2	Integer	
Befehlszeichen	126	127	2	Integer	
Anfangsdatum	128	129	2	Packed Rec.	
Monat					erste vier Bits
Tag					nächste fünf Bits
Jahr					letzte sieben Bits
Letzter Zugriff	130	131	2	Packed Record	
Monat					erste vier Bits
Tag					nächste fünf Bits
Jahr					letzte sieben Bits

Abb.1 Die im Textkopf einer Pascal-Textdatei gespeicherten Daten

Ich versuchte dann, die BLOCKREAD-Funktion zum Einlesen der Textkopfdaten zu verwenden, mit zufriedenstellendem Ergebnis. Da mit BLOCKREAD nur auf Einheiten, die mindestens einen vollen 512-Byte-Block umfassen, zugegriffen werden kann, mußte an die HEADER-Definition FILLER3 angehängt werden, um die Datenstruktur auf die erforderliche Größe zu bringen.

Nebenbei sei erwähnt, daß die Methode, mit der der Output wahlweise auf den Bildschirm oder auf den Drucker geschickt wird, auch in vielen anderen Anwendungen nützlich sein kann. Diese Aufgabe wird von der Prozedur WRITEINFO gelöst, die nur eine Datei, nämlich das als "interactive" deklarierte OUTPUTFILE, benutzt. Der dieser Prozedur übergebene Parameter ist ein String: "CONSOLE:" oder "PRINTER:" - damit ist es möglich, den Output auf das jeweils gewünschte Gerät zu schicken.

Für diejenigen Leser, die den Pascal-Editor häufig zur Textverarbeitung verwenden, kann ein Ausdruck der Environment-Informationen ihrer Textdateien (insbesondere der Kalenderdaten und Randbegrenzungen) recht nützlich sein.

Das Programm benötigt den Namen der Textdatei (das Suffix .TEXT wird automatisch vom Programm erzeugt) und die Angabe des Ausgabegerätes; der Output kann auf den Bildschirm, den Drucker oder auch auf beide Geräte geschickt werden. Beantwortet man die Frage nach der Textdatei oder dem Ausgabegerät mit einem einfachen Carriage-Return, wird das Programm vorzeitig beendet.

Die Ausgabe auf den Bildschirm oder den Drucker erfolgt im wesentlichen im gleichen Format. Abb.2 zeigt zwei Beispiele. Das erste Beispiel zeigt Daten, die aus der Kopfseite eines Pascal-Programms rekonstruiert wurden, in der weder die Umgebung (Environment) noch irgendwelche Marker festgesetzt wurden. In diesem Fall werden Standardparameterwerte eingetragen. Das zweite Beispiel zeigt Daten, die von einer Datei stammen, das ich zur Textverarbeitung benutzt hatte. Es handelt sich dabei, genauer gesagt, um diesen Artikel, den ich mit dem Pascal-Editor geschrieben habe. Einige der Umgebungsparameter wurden geändert und zwei Marker definiert.

```
              Text File: #5:tf.TEXT

    Auto Einr.= True        Fuellen = False
    Token def = True          Befehls-Z. = ^

    Linker Rand =  0        Recht. Rand   = 78
               Absatzrand   = 5

    Erstellungsdatum = 8-Mrz-82
    Letzte Aktualisierung = 13-Mai-85

          Marker Name     Markeradresse
               Keine Marker
```

Abb. 2 Bildschirm-/Drucker-Ausgabe des TEXT_INFO- Programms

Ich glaube, daß sich das Programm im großen und ganzen selbst erklärt. Ein paar Dinge bedürfen vielleicht noch einer näheren Erläuterung: Der Pascal-Record-Typ HEADER enthält die im ersten Block des Textkopfes gespeicherten Daten. FILLER1 und FILLER2 werden dazu benutzt, die nicht benötigten Bereiche zu überspringen. FILLER§ dient einzig und allein dem Zweck, mit Hilfe der BLOCKREAD-Funktion auf die Textkopf-Daten zuzugreifen. Der Grund für die Wahl dieser Funktion muß wohl noch extra erläutert werden.

Ich hatte versucht, die Daten in den HEADER-Record einzulesen, ohne die FILLER3-Definition (Länge 132 Bytes) zu benutzen, indem ich eine Datei vom Typ HEADER deklarierte und dann probierte, auf den ersten Record mit einem GET-Befehl zuzugreifen. Die in dieser Form gelesenen Daten stammten jedoch immer aus der zweiten Seite der Textdatei, also aus dem Bereich, der bereits den eigentlichen Text beinhaltet. Es hat den Anschein, daß das Betriebssystem eine Textdatei als solche erkannte, obwohl in der Deklaration als Dateityp HEADER und nicht TEXT angegeben wurde.

```
(*-------------------------------------------------------------*)
(*      LISTING#1:        TEXT_INFO                            *)
(*                                                             *)
(*      Dieses Programm untersucht den Dateikopf von Pascal Text *)
(*      Files und gibt die Fileparameter auf auf Drucker oder  *)
(*      Bildschirm aus.                                        *)
(*                                                             *)
(*      Programmiert von Alan J. Nayer          September 1981 *)
(*-------------------------------------------------------------*)

PROGRAM TEXT_INFO;

TYPE DATE=                        (* Format des Pascal Datums *)
       PACKED RECORD
         MONTH : 0..12;
         DAY   : 0..31;
         YEAR  : 0..100;
       END;

     HEADER=                      (* Format des Pascal Textkopf-Blocks *)
       PACKED RECORD
         FILLER1         : PACKED ARRAY[0..3] OF CHAR;
         MARKER          : PACKED ARRAY[0..9,0..7] OF CHAR;
         FILLER2         : PACKED ARRAY[0..9] OF CHAR;
         MARKER_ADDR     : PACKED ARRAY[0..9] OF INTEGER;
         AUTO_INDENT     : PACKED ARRAY[0..15] OF BOOLEAN;
         FILLING         : PACKED ARRAY[0..15] OF BOOLEAN;
         TOKEN_DEF       : PACKED ARRAY[0..15] OF BOOLEAN;
         LEFT_MARGIN     : INTEGER;
         RIGHT_MARGIN    : INTEGER;
         PARA_MARGIN     : INTEGER;
         COMMAND_CH      : CHAR;
         DATE_CREATED    : DATE;
         DATE_UPDATED    : DATE;
                              (* Mit FILLER3 wird der HEADER-Record *)
                              (* fuer BLOCKREAD auf die Groesse von *)
                              (* 512 Bytes gebracht.                *)
         FILLER3         : PACKED ARRAY[0..379] OF CHAR;
       END;

VAR TEXT_HDR             : HEADER;
    IN_FILE_NAME         : STRING;
    MONTH_NAMES          : STRING[36];
    CR,FF,CLEAR_EOL,BEL,NUL,DESTINATION
                         : CHAR;
    GOOD_DEST,SCREEN,PRINTER
                         : SET OF CHAR;
(*-----------------------------------------------------------------*)

PROCEDURE INITIALIZATION;

BEGIN
  CR:=CHR(13);
  FF:=CHR(12);
  CLEAR_EOL:=CHR(29);
  BEL:=CHR(7);
  NUL:=CHR(0);
  MONTH_NAMES:='JanFebMrzAprMaiJunJulAugSepOktNovDez';
  SCREEN:=['S','B','s','b'];
```

```
  PRINTER:=['P','B','p','b'];
  GOOD_DEST:=SCREEN+PRINTER
END; (*initialization*)

PROCEDURE GET_INPUT_FILE;

VAR TFILE         FILE;       (* Behandelt man ein Textfile als typfreies *)
                              (* File, dann kann man den Kopfblock lesen  *)
    NUM_BLOCKS  : INTEGER;
    GOOD_FILE   : BOOLEAN;

BEGIN
  WRITELN(FF,'TEXT INFO');
  REPEAT                     (* Bis Datei erfolgreich gelesen ist *)
   .GOTOXY(0,5);
    WRITE('Name des Textfiles (.TEXT wird angehaengt):',CR,CLEAR_EOL);
    READLN(IN_FILE_NAME);
    IF IN_FILE_NAME=' ' THEN EXIT(PROGRAM) ELSE
     (* Mit <RET> wird Programm verlassn *)
      IN_FILE_NAME:=CONCAT(IN_FILE_NAME,'.TEXT');
    GOTOXY(0,23);
                                                        (*$I-*)
    RESET(TFILE,IN_FILE_NAME);
    GOOD_FILE:=IORESULT=0;
    IF NOT GOOD_FILE THEN WRITE(IN_FILE_NAME,' nicht gefunden.',
                            CLEAR_EOL,BEL) ELSE
    BEGIN                  (* Wenn File erfolgreich eroeffnet, versuchen *)
                           (* ersten Kopfblock zu lesen.                 *)
      NUM_BLOCKS:=BLOCKREAD(TFILE,TEXT_HDR,1);
                                                        (*$I+*)
      IF (NUM_BLOCKS<>1) OR (IORESULT<>0) THEN
      BEGIN
        WRITE(IN_FILE_NAME,' nicht lesbar.',BEL,CLEAR_EOL);
        EXIT(PROGRAM)
      END ELSE WRITE(IN_FILE_NAME,'  geoeffnet und gelesen',CLEAR_EOL);
      CLOSE(TFILE,LOCK)
    END
  UNTIL GOOD_FILE
END;  (*get_input_file*)

PROCEDURE GET_OUTPUT_FILE;

BEGIN
  GOTOXY(0,10);
  WRITE  ('Ausgabe auf:');
  WRITELN('S(chirm, P)rinter oder B)eide');
  WRITELN;
  WRITE('Ihre Wahl: ');
  REPEAT
    READ(KEYBOARD,DESTINATION);
    IF EOLN(KEYBOARD) THEN EXIT(PROGRAM)
  UNTIL DESTINATION IN GOOD_DEST;
  WRITE(DESTINATION)
END;  (*get_output_file*)

PROCEDURE T_OR_F(BOOL:BOOLEAN; VAR TR_FL:STRING);

BEGIN
  IF BOOL THEN TR_FL:='True ' ELSE TR_FL:='False'
END;  (* t_or_f *)
```

```
PROCEDURE CONV_DATE(DATE_PARAM:DATE; VAR EXPANDED_DATE:STRING);

VAR S1,S2,S3      STRING[3];

BEGIN
  WITH DATE_PARAM DO
  BEGIN
    STR(DAY,S1);
    STR(YEAR,S3);
    S2:=COPY(MONTH_NAMES,MONTH*3-2,3);
    EXPANDED_DATE:=CONCAT(S1,'-',S2,'-',S3)
  END
END;  (*conv_date*)

PROCEDURE WRITE_INFO(OUT_FILE_NAME:STRING);

VAR OUTPUT_FILE           INTERACTIVE;  (* Diesem File wird entweder         *)
                                        (* CONSOLE: oder PRINTER: zugewiesen *)
    STRNG1,STRNG2         STRING;
    MARKER_NUM            INTEGER;
BEGIN
  REWRITE(OUTPUT_FILE,OUT_FILE_NAME);
  IF OUT_FILE_NAME='CONSOLE:' THEN WRITELN(OUTPUT_FILE,FF);
  WRITELN(OUTPUT_FILE,'Text File: ':20,IN_FILE_NAME,CR,CR);
  WITH TEXT_HDR DO
  BEGIN
    T_OR_F(AUTO_INDENT[0],STRNG1);
    T_OR_F(FILLING[0],STRNG2);
    WRITELN(OUTPUT_FILE,'Auto Einr.= ',STRNG1,
          'Fuellen = ':16,STRNG2);
    T_OR_F(TOKEN_DEF[0],STRNG1);
    WRITELN(OUTPUT_FILE,'Token def = ',STRNG1,
          'Befehls-Z. = ':22,COMMAND_CH,CR);
    WRITELN(OUTPUT_FILE,'Linker Rand = ',LEFT_MARGIN:2,
          'Recht. Rand  = ':22,RIGHT_MARGIN:2);
    WRITELN(OUTPUT_FILE,'Absatzrand  = ':26,PARA_MARGIN,CR);
    CONV_DATE(DATE_CREATED,STRNG1);
    CONV_DATE(DATE_UPDATED,STRNG2);
    WRITELN(OUTPUT_FILE,'Erstellungsdatum = ',STRNG1,CR,
          'Letzte Aktualisierung = ',STRNG2,CR);
    WRITE(OUTPUT_FILE,'Marker Name':16,'Markeradresse ':19);
                          (* Ist das erste Byte des Markernamens     *)
                          (* chr(0), dann ist der Marker leer.       *)
    IF MARKER[0,1]=NUL THEN WRITE(OUTPUT_FILE,CR,
                                  'Keine Marker    ':27) ELSE
      FOR MARKER_NUM:=0 TO 9 DO
        IF MARKER[MARKER_NUM,1]<>NUL THEN
          WRITE(OUTPUT_FILE,CR,MARKER_NUM:3,MARKER[MARKER_NUM]:12,
                MARKER_ADDR[MARKER_NUM]:14)
  END;
  IF OUT_FILE_NAME='PRINTER:' THEN WRITELN(OUTPUT_FILE,CR,CR)
END;  (* write_info *)

BEGIN  (* Hauptprogramm *)
  INITIALIZATION;
  GET_INPUT_FILE;
  GET_OUTPUT_FILE;
  IF DESTINATION IN SCREEN THEN WRITE_INFO('CONSOLE:');
  IF DESTINATION IN PRINTER THEN WRITE_INFO('PRINTER:');
END.
```

David Geddes

Das Pascal-Datum

Haben Sie schon einmal versucht, die Kalenderinformationen von Disketten zu aktualisieren, auf die mit einem Turnkey-System (SYSTEM.STARTUP) zugegriffen wird? Normalerweise muß man dazu den Filer aufrufen und das aktuelle Tagesdatum eingeben, bevor man das Programm benutzt. Die Verwendung eines Turnkey-Systems, also eines Programmsystems, das beim Einschalten des Rechners betriebsbereit ist, wird dadurch natürlich verhindert. Außerdem muß der Filer auf der Diskette vorhanden sein.

Als ich versuchte, dieses Problem zu bewältigen, suchte ich im Betriebssystem herum und schrieb bei dieser Gelegenheit die in sich abgeschlossene PROCEDURE GETDATE (s. Listing Nr.1). Das Programm DATEDEMO zeigt, wie man GETDATE in einem Turnkey-Programm benutzen kann. Wenn man sie in das SYSTEM.STARTUP-Programm einbaut, kann man mit ihr bei jedem Booten der Diskette das Tagesdatum eingeben.

GETDATE sieht zunächst nach dem auf der Diskette gespeicherten Datum und fragt, ob es geändert werden soll. Das neue Datum kann genau wie mit dem Filer eingegeben werden; der einzige Unterschied ist, daß man den Monat auch als Zahl angeben kann. Wenn der Benutzer eine Änderung eingibt, wird sowohl das Diskettendatum als auch das im Speicher befindliche Datum entsprechend aktualisiert, damit neu angelegte Dateien gleich das geänderte Datum erhalten. Das Datum (aktualisiert oder nicht) wird von GETDATE als variabler Parameter in Stringform übergeben (vgl. den Prozedurkopf von GETDATE). Die CONST DATELOC (Speicheradresse für das Datum) bezieht sich auf Version 1.1. Wenn Sie Version 1.0 verwenden, müssen Sie DATELOC in -22254 umändern.

Sie sollten vorsichtig sein, wenn Sie diese Prozedur zuerst ausprobieren. Testen Sie sie nur mit Disketten, die Sie vorher kopiert haben, da schlimme Dinge mit ihnen passieren können, wenn Sie die Prozedur nicht richtig abgetippt haben...

Solche Programmieraufgaben werden allerdings durch PASCAL-ZAP von Philip Ender (vgl. gleichnamiges Kapitel in diesem Buch) erleichtert.

```
(*******************************************)
(* Listing #1: Demoprogramm fuer GETDATE   *)
(*                                         *)
(* Von: David Geddes and Ron DeGroat       *)
(*      Mai 1981                           *)
(*******************************************)

PROGRAM DATEDEMO;

VAR DATE:STRING;

PROCEDURE GETDATE(VAR DATE:STRING);

(* Liest Datum und ermoeglicht Aenderung aehnlich *)
(* der D(ate-Option des SYSTEM.FILERs. Im Vari-   *)
(* ablenparameter wird das Datum in Form eines    *)
(* Strings zurueckgegeben.                        *)

CONST MONTHS='**JanFebMrzAprMaiJunJulAugSepOktNovDez';

      DATELOC=-21992;  (* -22254 fuer Version 1.0 *)

TYPE DATEREC = PACKED RECORD
                 MM : 0..12;
                 DD : 0..31;
                 YY : 0..99;
               END; (*daterec*)

     MEMDATEREC = RECORD CASE INTEGER OF
                    1:(DATE:^DATEREC);
                    2:(LOC : INTEGER);
                  END; (*memdaterec*)

VAR DISKBLK2:RECORD
               XXX     : PACKED ARRAY[0..19] OF CHAR;
               VOLDATE : DATEREC;
               ZZZ     : PACKED ARRAY[22..511] OF CHAR;
             END;

    NEWDATE      :DATEREC;
    MEMDATE      :MEMDATEREC;
    DD,MM,YY     :INTEGER;
    CHANGE       :STRING;
    DONE         :BOOLEAN;


FUNCTION INT(NUM:STRING):INTEGER;

(* Konvertiert String in Integerwert, funktioniert *)
(* nur bei positiven Werten.                       *)

VAR I,X:INTEGER;

BEGIN (*int*)
  X:=0;
  FOR I:=1 TO LENGTH(NUM) DO
    X:=10*X+(ORD(NUM[I])-ORD('0'));
  INT:=X;
END; (*int*)
```

```
PROCEDURE CHANGEDATE;

(* Aendert Datum entsprechend den Angaben des Benutzers *)

CONST UCMONTHS='**JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC';
      LCMONTHS ='**janfebmaraprmayjunjulaugsepoctnovdec';

VAR DASH:INTEGER;
    MONPART:STRING;
    NEWDATE:DATEREC;

PROCEDURE SETDATE;

(* Schreibt Datum auf Diskette und in Arbeitsspeicher *)

BEGIN
  DISKBLK2.VOLDATE:=NEWDATE;
  UNITWRITE(4,DISKBLK2,512,2,0);
  MEMDATE.LOC:=DATELOC;
  MEMDATE.DATE^:=NEWDATE;
END; (*setdate*)


BEGIN (*changedate*)

(* Tag aendern? *)

   DASH:=POS('-',CHANGE);
   CASE DASH OF
     0:DD:=INT(CHANGE);
     1:DD:=DISKBLK2.VOLDATE.DD;
     2,3:DD:=INT(COPY(CHANGE,1,DASH-1));
   END;
   IF (DD<1) OR (DD>31) OR (DASH>3)
   THEN NEWDATE.DD:=DISKBLK2.VOLDATE.DD
   ELSE NEWDATE.DD:=DD;
   IF DASH=0 THEN CHANGE:=''
   ELSE DELETE(CHANGE,1,DASH);

(* Monat aendern? *)

   DASH:=POS('-',CHANGE);
   IF DASH>0 THEN MONPART:=COPY(CHANGE,1,(DASH-1))
   ELSE IF LENGTH(CHANGE)>0 THEN MONPART:=CHANGE
        ELSE MONPART:='';
   CASE LENGTH(MONPART) OF
     0:MM:=DISKBLK2.VOLDATE.MM;
     1,2:MM:=INT(MONPART);
     3:MM:=POS(MONPART,MONTHS) DIV 3+POS(MONPART,UCMONTHS)
            DIV 3+POS(MONPART,LCMONTHS) DIV 3;
   END;
   IF (MM<1) OR (MM>12)
   THEN NEWDATE.MM:=DISKBLK2.VOLDATE.MM
   ELSE NEWDATE.MM:=MM;

(* Jahr aendern? *)

   IF (DASH>0) AND (LENGTH(CHANGE)>DASH) THEN
     BEGIN
```

```
        DELETE(CHANGE,1,DASH);
        YY:=INT(CHANGE)
      END
    ELSE YY:=DISKBLK2.VOLDATE.YY;
    IF (YY<0) OR (YY>99)
    THEN NEWDATE.YY:=DISKBLK2.VOLDATE.YY
    ELSE NEWDATE.YY:=YY;

    SETDATE; (* Auf Disk und im Speicher *)

END; (*changedate*)


PROCEDURE GETVOLDATE(VAR DATE:STRING);

VAR DAY,YEAR:STRING;

BEGIN
  UNITREAD(4,DISKBLK2,512,2,0);
  WITH DISKBLK2.VOLDATE DO
    BEGIN
      STR(DD,DAY);
      STR(YY,YEAR);
      DATE:=CONCAT(DAY,'-',COPY(MONTHS,3*MM,3),'-',YEAR);
    END;
END; (*getvoldate*)

 BEGIN (* Der Kern von GETDATE *)
   REPEAT
     GETVOLDATE(DATE);
     WRITELN('Heute ist der   ',DATE); (* Diskettendatum *)
     WRITE('Neues Datum? ');
     READLN(CHANGE);
     DONE:=(LENGTH(CHANGE)=0);
     IF NOT DONE THEN CHANGEDATE;
   UNTIL DONE;
   GETVOLDATE(DATE);
 END; (*getdate*)

 BEGIN (*datedemo*)
   GETDATE(DATE);
   WRITELN;
   WRITELN(DATE);
 END. (*datedemo*)
```

Chris Wilson

Verbesserte Tastatur-Routine

Wie Sie sicher wissen, gibt es einige ASCII-Zeichen, die mit der APPLE II-Tastatur nicht direkt erzeugt werden können. In der folgenden Tabelle sind alle sichtbaren Zeichen des ASCII-Zeichensatzes aufgelistet. Zeichen, die (im Gegensatz zu Kleinbuchstaben) nicht von der Tastatur aus eingegeben werden können, sind unterstrichen.

```
!"#$&'*()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO

PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
            --- --                         ----
```

Das in diesem Programm beschriebene Tastatur-Treiberprogramm ist eine Erweiterung des standardmäßigen Pascal 1.1-Tastaturtreibers. Mit dieser Erweiterung kann man nicht nur die zusätzlichen Zeichen direkt von der Tastatur aus eingeben, sondern auch die Kleinbuchstaben einfacher ansprechen als mit dem Standardtreiber.

Man kann die Programme ATTACHUD und SYSTEM.ATTACH des Pascal 1.1-BIOS-Softwarepakets benutzen, um einen solchen erweiterten Treiber zu installieren (vgl. Kapitel *Pascal intern* in diesem Buch).

Mein Programm ist eine Adaption des *Lazer Systems Input Editor* von Randy Hyde, der für die Funktion unter DOS und BASIC entwickelt wurde.

Das Programm filtert die Eingabezeichen heraus, die vom Standardtreiber gelesen werden. Schreiboperationen, Initialisierung und Statusaufrufe gehen direkt an den Standardtreiber.

Zur Steuerung des erweiterten Treibers werden Escape-Sequenzen verwendet. Eine Reihe von Escape-Sequenzen steuert die Eingabe von Sonder-

zeichen, während die restlichen zur Eingabe von Kleinbuchstaben verwendet werden können.

Da der Standardtreiber weiterhin dazu dient, die eigentliche Tastaturabfrage zu erledigen und den Eingabepuffer abzuarbeiten, muß die Eingabemöglichkeit für Kleinbuchstaben über den Standardtreiber mit Ctrl-E eingeschaltet werden. Beachten Sie, daß Ctrl-E, Ctrl-W, Ctrl-R und Ctrl-T zur Steuerung des Standardtreibers benutzt werden.

Der erweiterte Treiber erlaubt die Verwendung einer Software-Shift-Taste und einer Shift-lock-Taste. Man beginnt mit dem Eingabemodus für Kleinbuchstaben, sobald man Ctrl-E wie oben erwähnt eingibt. Will man jetzt einen Großbuchstaben eintippen, braucht man nur vorher die ESC-Taste zu drücken. Ein großes "A" wird also beispielsweise durch die Tastenfolge "ESC a" erzeugt (entsprechend dem "Shift a" auf einer Schreibmaschine). Das geht einfacher als mit dem aus zwei Tasten bestehenden "Ctrl-W"-Präfix, das man zur Eingabe von Großbuchstaben verwenden muß, wenn man den Standardtreiber benutzt. Will man den Shift-lock-Modus benutzen, also fortlaufend Großbuchstaben eingeben, muß man "ESC ESC" tippen. Zum Umschalten auf den Kleinbuchstabenmodus wird dann später wieder "ESC ESC" eingegeben.

Soviel zur Eingabe von Kleinbuchstaben. Die Escape-Sequenzen werden aber auch für die Eingabe gewisser Sonderzeichen benötigt. Hier eine Tabelle dieser Zeichen:

```
ESC 1 or !  -> |    (*chr(124)*)
ESC 2 or "  ->      (*chr(126)*)
ESC 3 or #  -> del  (*chr(127)*)
ESC 7 or '  -> `    (*chr(96)*)
ESC 8 or (  -> {    (*chr(123)*)
ESC 9 or )  -> }    (*chr(125)*)
ESC , or <  -> [    (*chr(91)*)
ESC . or >  -> ]    (*chr(93)*)
ESC - or =  -> _    (*chr(95)*)
ESC / or ?  -> \    (*chr(92)*)
ESC <space> -> ESC
```

Da das ESC-Zeichen zur Eingabe dieser Sonderzeichen benötigt wird, wurde die Möglichkeit geschaffen, das ESC-Zeichen selbst einzugeben, um Programme wie z.B. den Editor zu bedienen. Sie brauchen dazu nur "ESC «Leertaste»" zu tippen.

Wenn Sie ATTACHUD starten, um das Datenfile für SYSTEM.ATTACH zu erzeugen, beantworten Sie seine Fragen wie folgt:

Enter name of attach data file:
(Name des Anpassungs-Datenfiles:)
ATTACH.DATA

Will you ever use the (2000,3FFF hex) HI-Res page?
(Wird HI-RES-Grafikseite 1 ($2000 bis $3FFF) benutzt?)
N

Will you ever use the (4000,5FFF hex) HI-Res page?
(Wird HI-RES-Grafikseite 2 ($4000 bis $5FFF) benutzt?)
N

What is the name of this driver?
(Wie heißt dieses Treiberprogramm?)
Hier muß der .PROC-Name des Assembler-Quellfiles angegeben werden.
Bei Eingabe von «Return» wird das Programm verlassen:
CONSOLE

Which Unit numbers should refer to this device driver?
(Welche Gerätenummern sollen sich auf diesen Treiber beziehen?)

Unit number (bei «Return» Programmabbruch):
1

Do you want this unit to be initialized at boot time?
(Soll dieses Gerät beim Booten initialisiert werden?)
Y

Do you want another unit number to refer to this device driver?
(Soll sich noch eine andere Gerätenummer auf diesen Treiber beziehen?)
Y

Unit number (bei «Return» Programmabbruch):
2

Do you want this unit to be initialized at boot time?
(Soll dieses Gerät beim Booten initialisiert werden?)
N

Do you want another unit number to refer to this device driver?
(Soll sich noch eine andere Gerätenummer auf diesen Treiber beziehen?)
N

Do you want this driver to start on a certain byte boundary?
(Soll dieser Treiber bei einer bestimmten Byte-Grenze beginnen?)
N

Do you want to attach another driver?
(Soll noch ein Treiber angepaßt werden?)
N

```
; Copyright (c) 1982 Chris Wilson
;
; Die von diesem erweiterten Tastaturtreiber
; unterstuetzten Escape-Sequenzen lauten
; wie folgt :
;
;   ESC 1 or !  -> |    (*chr(124)*)
;   ESC 2 or "  -> ~    (*chr(126)*)
;   ESC 3 or #  -> del  (*chr(127)*)
;   ESC 7 or '  -> `    (*chr(96)*)
;   ESC 8 or (  -> {    (*chr(123)*)
;   ESC 9 or )  -> }    (*chr(125)*)
;   ESC , or <  -> [    (*chr(91)*)
;   ESC . or >  -> ]    (*chr(93)*)
;   ESC - or =  -> _    (*chr(95)*)
;   ESC / or ?  -> \    (*chr(92)*)
;   ESC <space> -> ESC
;   ESC ESC     -> (* Umschalter fuer Grossbuchstaben-Feststelltaste *)
;   ESC a       -> A
;    :  :       -> :
;   ESC z       -> Z
;
; Zur Installation dieses Treibers wird das Programm
; SYSTEM.ATTACH benoetigt
;
;

INDIRECT .EQU  002
JVAFOLD  .EQU  OEE
ACJVAFLD .EQU  OE2

         .PROC CONSOLE

         JMP   CONCKHDL ;SYSTEM.ATTACH patcht die CONCK-Routine so,
                        ;dass die Einsprungadresse hier liegt
         STA   TEMP1    ;Einsprungadresse fuer alle Lese-, Schreib-
                        ;Initialisierungs- und Statusaufrufe
         STY   TEMP1+1
         PLA            ;Ruecksprungadresse retten
         STA   RETURN
         PLA
         STA   RETURN+1
         TXA            ;X-Register wird zur Ermittlung des Aufruf-
                        ;typs verwendet
         BEQ   READ
         CMP   #1
         BEQ   WRITE
         CMP   #2
         BEQ   INIT
         CMP   #4
         BEQ   STATUS
         LDX   #3       ;E/A-Befehl existiert nicht
         JMP   RET

RETURN   .WORD 0
TEMP1    .WORD 0
ROUTINE  .WORD 0
```

```
READ    ;Code fuer Leseoperation
        LDA   RDRTN+1   ;Stackadresse festhalten, damit die Standard-
        PHA             ;Leseroutone zu READRTN zurueckspringt und man
                        ;auf diese Weise die Tastaturzeichen ausfiltern
                        ;kann
        LDA   RDRTN
        PHA
        LDY   #1        ;Adressoffset der Standard-Leseroutine
        BNE   GET1      ;in der von SYSTEM.ATTACH erzeugten Sprung-
                        ;vektorkopie

WRITE   ;Code fuer Schreiboperation
        LDY   #4
        BNE   GET

INIT    ;Code fuer Initialisierungsoperation

        ;Zuerst BIOS-Sprungvektor so patchen, dass alle Schreib-,
        ;Initialisierungs- und Statusaufrufe direkt an die Stan-
        ;dardkonsolenroutinen geleitet werden.

        LDA   0C08B     ;Auf Interpreter umschalten
        LDY   #4
        JSR   FIXUP     ;Schreiben
        LDY   #7
        JSR   FIXUP     ;Initialisieren
        LDY   #43.
        JSR   FIXUP     ;Status

        ;Jetzt den von SYSTEM.ATTACH erzeugten Patch aufheben,
        ;der einen Sprung an den Anfang dieses Treiberprogramms
        ;bewirkt.

        LDY   #55.      ;Original CONCK-Adresse holen
        LDA   @ACJVAFLD,Y
        STA   INDIRECT
        INY
        LDA   @ACJVAFLD,Y
        STA   INDIRECT+1
        LDY   #0
        LDA   #08       ;PHP
        STA   @INDIRECT,Y
        INY
        LDA   #48       ;PHA
        STA   @INDIRECT,Y
        INY
        LDA   #8A       ;TXA
        STA   @INDIRECT,Y
        LDA   0C083     ;BIOS einschalten

        LDY   #7
        BNE   GET

STATUS  ;Code fuer Status
        LDY   #43.

GET     LDA RETURN+1    ;Ruecksprungadresse auf Stack bringen, damit
                        ;der Standardtreiber direkt zum aufrufenden
                        ;Programm zurueckspringt
```

```
        PHA
        LDA RETURN
        PHA

GET1    ;ACJVAFLD enthaelt einen Zeiger auf die von
        ;SYSTEM.ATTACH angelegte Kopie des Sprungvektors,
        ;die erzeugt wurde, bevor er an diesen Treiber
        ;angepasst wurde.
        LDA   @ACJVAFLD,Y
        STA   ROUTINE
        INY
        LDA   @ACJVAFLD,Y
        STA   ROUTINE+1
        LDY   TEMP1+1  ;Register wiederherstellen
        LDA   TEMP1
        JMP   @ROUTINE ;Standardtreiber aufrufen

FIXUP   LDA   @ACJVAFLD,Y
        STA   @JVAFOLD,Y
        INY
        LDA   @ACJVAFLD,Y
        STA   @JVAFOLD,Y
        RTS

RDRTN   .WORD READRTN-1
ESRTN   .WORD ESCRTN-1
CAPSLOCK .BYTE 0

TABLE1  .ASCII "123789,.-/"
        .ASCII "!"
        .BYTE  022
        .ASCII "#'()<>=?"
        .ASCII " "
TABLE2  .BYTE  07C,07E,07F,060,07B,07D,05B,05D,05F,05C
        .BYTE  07C,07E,07F,060,07B,07D,05B,05D,05F,05C
        .BYTE  01B
TBLSIZE .EQU  *-TABLE2

READRTN CMP   #01B      ;ESC?
        BEQ   ESCKEY
        LDY   CAPSLOCK ;Capslock gesetzt?
        BNE   CHKLC
        CMP   #041      ;Nein, GROSSBUCHSTABENMODUS?
        BCC   RET
        CMP   #05A+1
        BCS   RET
        ORA   #020      ;Ja, Kleinbuchstaben generieren
        JMP   RET
CHKLC   CMP   #061      ;Kleinbuchstabe?
        BCC   RET
        CMP   #07A+1
        BCS   RET
        AND   #0DF      ;Ja, in Grossbuchstaben umwandeln
        JMP   RET
ESCKEY  LDA   ESRTN+1   ;ESCRTN auf Stack bringen, damit die Standard-
        PHA             ;Leseroutine nach ESCRTN zurueckspringt
        LDA   ESRTN
        PHA
        LDY   #1
```

```
        JMP     GET1

ESCRTN  LDY     #TBLSIZE
CONVLOOP CMP    TABLE1,Y ;Zeichen nach ESC untersuchen, um festzustellen,
        BEQ     CHANGE   ;ob es eines der besonderen Steuerzeichen ist
        DEY
        BPL     CONVLOOP
        JMP     NOTSPCL
CHANGE  LDA     TABLE2,Y ;Ja, ersetze es durch das entsprechende Zeichen
        JMP     RET      ;aus TABLE2

NOTSPCL CMP     #01B     ;ESC?
        BNE     NOTESC
        LDA     CAPSLOCK ;Ja, Capslock umschalten
        EOR     #0FF
        STA     CAPSLOCK
        JMP     READ     ;Naechstes Zeichen holen

NOTESC  CMP     #061     ;Kleinbuchstabe?
        BCC     RET
        CMP     #07A+1
        BCS     RET
        AND     #0DF     ;Ja, in Grossbuchstaben umwandeln

RET     TAY              ;Akku retten
        LDA     RETURN+1 ;Ruecksprungadresse auf Stack, damit
        PHA              ;zur aufrufenden Routine zurueckgesprungen wird
        LDA     RETURN
        PHA
        TYA              ;Akku wiederherstellen
        RTS

CONCKHDL PHP             ;Prozessorstatus wiederherstellen
        PHA
        TXA
        PHA
        TYA
        PHA
        CLC              ;CONCK-Adresse holen
        LDY     #55.
        LDA     @ACJVAFLD,Y
        ADC     #6       ;Wegen des von SYSTEM.ATTACH erzeugten Patches
                         ;um 6 erhoehen
        STA     ROUTINE
        INY
        LDA     @ACJVAFLD,Y
        ADC     #0
        STA     ROUTINE+1
        JMP     @ROUTINE

        .END
```

Paul W. Mosher

Strukturierter Zugriff auf das DOS-Directory

Bei meinem Bemühungen, mich mit Pascal besser zurechtzufinden, war es nicht leicht, mit den Programmiermethoden zu brechen, die ich mir im Laufe der Jahre beim Umgang mit BASIC angewöhnt hatte. Wenn ich mir heute meine ersten Pascal-Programme ansehe, muß ich zugeben, daß viele von ihnen praktisch wörtliche Übersetzungen von Ideen waren, die ich in BASIC entwickelt hatte. Besonders das Konzept der "abstrakten Datentypen" hatte es bei mir anfänglich schwer, in meine Sammlung nützlicher Programmiertechniken aufgenommen zu werden, obwohl gerade dieses Konzept einen der Hauptvorteile von Pascal und ähnlichen Sprachen gegenüber den altmodischeren BASIC-artigen Sprachen darstellt. Dieses Problem stellte sich aber offensichtlich nicht nur mir, denn viele Pascal-Programme und -Prozeduren, die bisher veröffentlicht wurden, scheinen abstrakte Datentypen ebenso zu vermeiden.

Als Übung für mich selbst und auch als möglicher Grundstock für die Entwicklung von in Pascal geschriebenen DOS 3.3-Hilfsprogrammen probierte ich aus, wie man das VTOC (Volume Table Of Contents; Verzeichnis der Sektorbelegung) einer DOS 3.3-Diskette mit Hilfe abstrakter Datentypen darstellen kann. Beachten Sie, daß man das Ergebnis des hier abgedruckten Programms auch ohne abstrakte Datentypen erzielen kann, so als ob man in BASIC programmieren würde. Die Verwendung von Recordstrukturen macht jedoch das Programm viel besser verständlich und wahrscheinlich auch wesentlich kürzer. Mit neun kurzen Zeilen wird ein Diskettenverzeichnis ausgegeben, das freie und belegte Sektoren angibt, und mit einer einzigen Anweisung rechnet das Programm aus, wie viele Sektoren auf der Diskette noch frei sind.

Ein Stück Wirklichkeit

Abstrakte Datentypen werden mit dem Ziel verwendet, eine abstrakte Darstellung eines Ausschnitts der "Wirklichkeit" zu erreichen. Im Idealfall werden die Daten im Programm so beschrieben, daß die "Bedeutung" jedem klar wird, der das Programm liest. Im Falle dieses Kapitels besteht die "Wirklichkeit" aus der Folge von 256 Bytes, die in Sektor 0 auf Spur 17 jeder DOS 3.3-Diskette zu finden sind. Zum besseren Verständnis meiner Beschreibung des strukturierten Records lesen Sie am besten die Seiten 132 bis 133 des DOS 3.3(oder 3.2)-Handbuches.

Die dort abgebildete Tabelle zeigt Ihnen die Bedeutung der einzelnen Bytes des VTOC-Sektors. Wie man sofort sieht, werden mit diesen Bytes sehr unterschiedliche Sachverhalte dargestellt. Die Bytes $1 und $2 stellen zum Beispiel jeweils eine einzelne Zahl dar, die die Spur bzw. den Sektor angibt, bei dem der Diskettenkatalog beginnt. Im Gegensatz dazu werden die Bytes $36 und $37 zusammen als eine einzige Zahl behandelt, die die Byte-Anzahl in jedem Diskettensektor angibt.

Die Bytes $30 bis $33 muß man sich als ein Array aus 32 einzelnen Bits vorstellen. Im weiteren Verlauf dieser Tabelle findet man dieses aus 32 Bits bestehende Array noch 34 mal (140 Bytes). Es bildet damit die Disk Bit Map (Bytes $38 bis $C5). Jedes einzelne Bit des insgesamt 1120 Bit langen Arrays stellt einen einzelnen Diskettensektor dar (die Hälfte wird freigelassen). Eine Eins zeigt dabei jeweils an, daß der Sektor frei ist, während eine Null einen belegten Sektor bezeichnet. Es wäre - gelinde gesagt - unklug, diese 140 Bytes als Dezimal- oder Hexadezimalzahlen anzusprechen, da ihre "Bedeutung" in den einzelnen Bits besteht. Aus dem gleichen Grund gibt es keinen Anlaß, die Bytes $36 und $37 einzeln anzusprechen, da ihre "Bedeutung" in einer einzelnen Zahl besteht. Mit strukturierten Records versucht man nun, solche Sachverhalte aus der "Wirklichkeit" möglichst genau nachzubilden.

Wie man dem DOS-Handbuch entnehmen kann, wird die Angelegenheit dadurch kompliziert, daß es eine Reihe von Bytes gibt, die "nicht benutzt" werden und über den Sektor verstreut sind. In einem strukturierten Record, in dem diese 256 Bytes als VTOC eingelesen und abgebildet werden sollen, müssen diese unbenutzten Bytes berücksichtigt werden, damit alles an seinen richtigen Platz kommt.

Darstellung des VTOC als abstrakten Datentyp

Damit die Beschreibung besser verständlich wird, habe ich das Pascal-Programm mit Zeilennummern aufgelistet, auf die ich mich im folgenden beziehe (vgl. Listing). Die Aufgabe besteht darin, die 256 Bytes mit Hilfe von

Variablen (Komponenten) zu beschreiben, die durch ihren Namen und TYPE die Bedeutung des VTOC wiedergeben.

Wir beginnen (Zeile 9) mit der Definition des Typs "byte" als Integer-Zahl im Bereich 0..255 (Untermenge). Ein aus solchen "Bytes" bestehendes Array würde tatsächlich jeweils zwei Speicher-Bytes je Element belegen, da Integer-Zahlen unabhängig vom zugelassenen Wertebereich in zwei Bytes gespeichert werden. Das höherwertige Byte würde in diesem Fall nicht benutzt werden. Verwendet man jedoch ein PACKED ARRAY oder einen PACKED RECORD, dann sorgt das Pascal-Betriebssystem dafür, daß die Variablen eines gegebenen Typs so wenig Speicherplatz wie möglich beanspruchen. In unserem Fall belegt damit eine Variable vom Typ "byte" genau ein Hauptspeicher-Byte.

Mit der Definition eines gepackten Records namens "VTOC_structure" in den Zeilen 16 bis 34 belegt eine Variable vom Typ "byte" genau eine Speicherzelle. Der Record beginnt mit Unused_A (Zeile 17), womit das nicht verwendete Byte Nr.$0 am Anfang des VTOC-Sektors dargestellt wird. Die nächsten drei Bytes bestehen aus echten Integer-Zahlen des Typs "byte" und haben eigene Namen mit entsprechender Bedeutung (Zeilen 18 bis 20).

Die beiden nicht benutzten Bytes Nr.$4 und $5 werden durch ein gepacktes Byte-Array dargestellt, das natürlich ebenfalls aus zwei Bytes besteht. Ich habe mich dazu entschlossen, sie entsprechend ihrer Position im Record zu nummerieren; ich hätte aber das Array genauso gut mit der Dimensionierung [0..1] wählen können. Beachten Sie, daß dieses gepackte Array mit einer geraden Byte-Nummer im Record beginnt. Ich erwähne das nur, weil das Pascal-Betriebssystem mit "Worten", d.h. Zwei-Byte-Gruppen, arbeitet und alle gepackten Arrays bei einer Wortgrenze beginnen müssen, auch wenn das bedeutet, daß man ein Byte verschenkt, wenn man eine gepackte Datenstruktur in einer anderen deklariert (vgl. *Pascal Language Reference Manual*, S.17,18).

Das nächste Byte, $6, beinhaltet die Diskettennummer und hat einen entsprechenden Namen (s. Zeile 22). Auf diese Nummer folgt eine Reihe unbenutzter Bytes ($7 bis $26). Ich hatte erst versucht, sie in einem "Packed Array [0..31] of Byte" unterzubringen, aber das ging aus dem gerade genannten Grund schief. Das gepackte Array begann tatsächlich bei Byte Nr.$8; Byte Nr.$7 blieb frei, was zur Folge hatte, daß sich die nachfolgenden Speicherplätze um ein Byte verschoben. Ich ging das Problem dann an, indem ich Byte Nr.$7 in Zeile 23 einzeln füllte und den Rest mit einem gepackten Array auffüllte. Das wollte aber ebenfalls nicht klappen.

Wenn man ein gepacktes Array mit einer ungeraden Byte-Anzahl deklariert und bei einer Wortgrenze beginnt, dann endet das Array in der Mitte eines Wortes. Das Betriebssystem weist dann den Rest *dieses* Wortes dem Array zu und ruft damit ein ähnliches Problem hervor wie oben beschrieben

(vgl. *APPLE Pascal Language Reference Manual*, S.16). Des Rätsels Lösung besteht darin (Zeilen 24 und 25), das Array nur mit 30 Bytes Länge und das 32. Byte getrennt zu deklarieren. Von hier an bis $30 gibt es dann keine weiteren Schwierigkeiten.

Die bei Position $30 beginnenden vier Bytes enthalten 32 einzelne Bits, und nicht etwa Zahlen. Wir brauchen hier also eine andere Methode, vier Bytes zu beschreiben, mit der man gleichzeitig auf 32 Bits zugreifen kann.

Die "Bit Map" für jeden Sektor einer Diskettenspur enthält ein Bit pro Sektor plus 16 unbenutzte Bits (vgl. S.133 unten des DOS-Handbuches). Alle Bit Maps zusammen ergeben die Bit Map für die gesamte Diskette. Ich definierte ein einzelnes Sector_bit (Zeile 10) als einen Variablentyp, der nur zwei mögliche Werte annehmen kann: "In_use" und "Free". Listet man die Namen in dieser Reihenfolge (und nicht etwa: Free, In_use) auf, so wird der Wert 0 mit "In_use" und der Wert 1 mit dem Namen "Free" verknüpft (so wie das auch auf der BASIC-Diskette der Fall ist).

Jetzt kann man (in Zeile 11) die Bit Map einer ganzen Spur, die "Track_bit_map", als ein Array aus 32 Sector_bits definieren. Genau wie der Typ "Boolean" kann der "Sector_bit"-Datentyp nur zwei Werte annehmen, so daß er in einem gepackten Array nur ein Speicherbit Platz benötigt. Das gesamte Array aus 32 Sector_bits kann so als "Packed Array [0..31] of Sector_bit" deklariert werden, was der Darstellung auf der BASIC-Diskette sehr nahe kommt.

Leider sind die DOS-VTOC-Bytes in der falschen Reihenfolge angeordnet, so daß das durch "Track_bit_map[6]" dargestellte Bit in Wirklichkeit Sektor Nr.14 repräsentiert usw. (vgl. DOS-Handbuch S.133 unten). Ich habe dieses Problem in meinem Programm dadurch gelöst, daß ich eine Abbildungs-FUNCTION implementierte, mit deren Hilfe ich auf die richtigen Sektoradressen schließen konnte. Wenn es eine natürlichere oder "ästhetischere" Deklaration für diese Record-Komponente gibt, bei der die Bits in der richtigen Reihenfolge dargestellt werden, dann habe ich sie nicht gefunden. Ich bin daher jedem dankbar, der hierfür eine bessere Lösung anbieten kann. Das Problem stellt sich wie folgt dar:

	Erste vier Bytes	Folgende vier Bytes
Arrayposition :	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Dargestellter Sektor :	8 9 10 11 12 13 14 15	0 1 2 3 4 5 6 7

Die Bits 16 bis 31, die in Byte 3 und 4 stehen, werden natürlich nicht benutzt. In Zeile 28 wird der Typ "Track_bit_map" verwendet, um vier Bytes zu belegen.

Die Bytes $34 und $35 in der Beschreibung des DOS-Handbuchs lassen sich ohne Schwierigkeiten abbilden und werden in den Zeilen 29 und 30 dargestellt. Die Bytes $36 und $37 sind als einzelne Integer-Zahl zu behandeln und bereits in der Reihenfolge gespeichert, in der auch das Pascal-System mit Integer-Zahlen umgeht. Aus diesem Grund kann in Zeile 31 die Komponente "Bytes_per_sec" als Integer-Zahl definiert werden.

Bei Byte $38 des Diskettensektors beginnt das Sektorverzeichnis, das die nächsten 140 Bytes belegt und angibt, welche Sektoren benutzt sind. Dementsprechend wird in Zeile 12 der Datentyp "Disk_bit_map" als gepacktes Array aus 35 "Track_bit_maps" deklariert. Mit diesem Datentyp werden in Zeile 32 die gesamten 140 Bytes des Sektorverzeichnisses definiert.

Variablen mit dem neuen Datentyp

Nun, da der Datentyp "VTOC_structure" deklariert ist, kann man eine Variable dieses Typs definieren (Zeile 42). Diese Variable (VTOC) hat eine Größe von 256 Bytes, in die man nun das VTOC einer DOS 3.3-Diskette einlesen kann. Die Komponenten von VTOC werden dann einfach über ihre Namen angesprochen (z.B. "VTOC.Volume_number"), ohne daß man irgendwelche Programmiertricks anwenden muß! Mit dem Aufruf der Funktion"VTOC.bit_map [track, sector]" können Sie sogar ein einzelnes Bit ansprechen. Das wäre allerdings aufgrund der genannten Probleme nicht das richtige Bit. Für den korrekten Zugriff benutze ich daher die Funktion "Mapping_of" in den Zeilen 47 bis 52. Durch einen Zugriff wie z.B. "VTOC.bit_map [track, mapping_of (sector)]" erhält man nun das richtige Bit.

Die Verwendung strukturierter Datentypen

Das Programm selbst ist nur ein Beispiel für die Benutzung der VTOC-Variablen. In den vier Zeilen 58, 59, 65 und 66 wird das VTOC der DOS 3.3-Diskette in die VTOC-Variable eingelesen, indem zunächst der Block mit Spur 17, Sektor 0 in einem Puffer gespeichert wird (Zeile 65), woraufhin die ersten 256 Bytes des Puffers mit MOVELEFT in die Variable VTOC kopiert werden (Zeile 66). In den Zeilen 89 bis 98 werden die belegten Sektoren der Diskette ausgegeben, während in den Zeilen 103 bis 108 die freien Sektoren der Diskette gezählt und angezeigt werden.

```
Program VTOC_READ;

CONST
  VTOC_block    = 136;  (* Untere Haelfte dieses Blocks ist VTOC Sektor *)

TYPE
  Sector        = 0..15;
  Track         = 0..34;
  Byte          = 0..255;
  Sector_bit    = (In_use, free);
  Track_bit_map = Packed Array[0..31] of Sector_bit;
  Disk_bit_map  = Packed Array[0..34] of Track_bit_map;
  Blockbuffer   = Packed Array[0..511] of byte;


  VTOC_structure= Packed Record
           Unused_A      :    Byte;
           Dir_sec_start :    Byte;
           Dir_Trk_start :    Byte;
           Dos_Release   :    Byte;
           Unused_B      :    Packed Array [4..5] of byte;
           Volume_number :    Byte;
           Unused_C      :    Byte;
           Unused_D      :    Packed array [8..37] of byte;
           Unused_E      :    Byte;
           Max_TS_Pairs  :    Byte;
           Unused_F      :    Packed array [40..47] of byte;
           Mask_bytes    :    Track_bit_map;
           Tracks_on_disk:    Byte;
           Secs_per_track:    Byte;
           Bytes_per_sec :    Integer;
           Bit_map       :    Disk_bit_map;
           Unused_G      :    Packed array [196..255] of byte;
   End;  (* VTOC-Struktur *)


VAR
  Unit_num,
  Block,
  Free_spaces       Integer;
  Dummy             Char;
  VTOC              VTOC_structure;
  Input_buffer      Block_buffer;
  Sec_num           Sector;
  Trk_num           Track;

    Function Mapping_of (To_be_mapped : Sector) : Sector;
        Begin
          If To_be_mapped < 8
             then Mapping_of :=  To_be_mapped + 8
             else Mapping_of :=  To_be_mapped - 8
        End;


Begin  (* Hauptprogramm *)

  Unit_num := 5;  (* Ich habe Laufwerk 2 in Slot 6 fuer die DOS-Diskette
                     benutzt. *)
  Block    := VTOC_block;
```

```
(* ---------------------------------------------------------------
   Lies den Block mit dem VTOC-Sektor in den Eingangspuffer und kopiere
   dann die ersten 256 Bytes des Eingangspuffers in die Variable VTOC
   --------------------------------------------------------------- *)
  Unitread (Unit_num, Input_buffer, 512, Block);
  Moveleft (Input_buffer[0], VTOC, 256);


(* ---------------------------------------------------------------
   Ausgabe diverser VTOC-Informationen
   --------------------------------------------------------------- *)
  Writeln('Directory beginnt bei Sekt.:', VTOC.Dir_sec_start );
  Writeln('Directory beginnt bei Spur :', VTOC.Dir_trk_start );
  Writeln('DOS Version                :', VTOC.DOS_release   );
  Writeln('Diskettennummer ist        :', VTOC.Volume_number );
  Writeln('Groesste S/S Paarzahl in S.:', VTOC.Max_TS_pairs  );
  Writeln('Anzahl der Diskettenspuren :', VTOC.Tracks_on_disk);
  Writeln('Anzahl der Sektoren/Spur   :', VTOC.Secs_per_track);
  Writeln('Bytes/Sektor               :', VTOC.Bytes_per_sec );
  Writeln('Groesse der "VTOC"-Variab. :', SIZEOF(VTOC)       );
  Writeln;
  Write(' << Belegungsplan und freier Platz auf Tastendruck >>');
  Read(Dummy); Writeln;


(* ---------------------------------------------------------------
   Tabellendarstellung der Diskette, aus der die Sektorbenutzung her-
   vorgeht
   --------------------------------------------------------------- *)
  For Sec_num := 0 to 15 do
    begin
     Write(Sec_num:2,': ');
     For Trk_num := 0 to 34 do
       If VTOC.bit_map[ Trk_num, Mapping_of(Sec_num) ] = free
           then write('.')
           else write('*');
       Writeln;
    end; (* For sec_num - Schleife *)
  Write('    ');For Trk_num := 0 to 34 do write (Trk_num MOD 10);Writeln;

(* ---------------------------------------------------------------
   Berechnung und Ausgabe der freien Diskettensektoren
   --------------------------------------------------------------- *)
  Free_spaces := 0;
  For Trk_num := 0 to 34 do
    for Sec_num := 0 to 15 do
      if VTOC.bit_map[Trk_num, Sec_num] = free
        then Free_spaces := Free_spaces + 1;
  Writeln ('    Freier Diskettenplatz = ', Free_spaces, ' Sektoren');

End.
```

Roy Bollinger

Terminal-unabhängige Bildschirmsteuerung

Ich bin seit über 18 Jahren im Bereich Software-Entwicklung tätig und habe mich insbesondere der Verbesserung von Software-Schnittstellen gewidmet. Ich konnte mich daher nicht damit zufrieden geben, daß meinem neuen APPLE-Pascal-System die Möglichkeiten zur vollständigen Kontrolle des Bildschirmcursors fehlten. Da ich fest davon überzeugt bin, daß der APPLE ein exzellenter Rechner ist, wußte ich, daß eine vollständige Cursorsteuerung möglich sein mußte - es galt nur noch herauszufinden, wie.

Beim Lesen des *APPLE-Pascal-Handbuches* fand ich einen Abschnitt mit der Überschrift:

4.3 SYSTEM RECONFIGURATION: SETUP.CODE

der auf Seite 240 beginnt. Da die Informationen, die ich dort fand, meiner Einschätzung nach alles enthielten, um den APPLE unabhängig von einem bestimmten Terminal betreiben zu können, beschloß ich, dort mit meinen Nachforschungen zu beginnen. Innerhalb des Abschnitts fand ich auf S.246 einen Unterabschnitt mit dem Titel:

4.3.4 VIDEO SCREEN CONTROL CHARACTERS

Damit war ich beim Kern der Sache, denn hier fand ich alle Bildschirm-Kontrollzeichen, die ich suchte.

Zunächst benutzte ich ein File-Dump-Programm, um die MISCINFO-Datei auszudrucken, das von dem Programm SETUP.CODE erzeugt wird. Danach startete ich SETUP.CODE und beantwortete jede der Fragen, die eine numerische Eingabe erfordern, mit aufeinanderfolgenden, positiven Integer-Werten, wobei ich bei eins begann. Das so erzeugte MISCINFO-File

ließ ich dann durch das Dump-Programm ausdrucken und verglich die Ergebnisse mit dem Original-Dump. So konnte ich feststellen, an welcher Stelle SETUP.CODE die Antworten in MISCINFO eingetragen hatte. In Tabelle 1 finden Sie das Ergebnis dieses Vergleiches.

Danach benutzte ich SETUP.CODE mehrere Male, um die Fragen zu beantworten, die die Antworten TRUE/FALSE verlangen. So konnte ich feststellen, wo und wie die Boole'schen Variablen in MISCINFO gespeichert wurden. Das Ergebnis dieser Arbeit finden Sie ebenfalls in Tabelle 1.

Mit diesen Informationen bewaffnet entschloß ich mich, eine PASCAL-Unit zu schreiben, die dem Programmierer die Möglichkeit gibt, den Bildschirmcursor vollständig zu kontrollieren. Diese Unit ist in Listing Nr.1 abgedruckt.

Der Vorteil dieser Unit liegt darin, daß Sie mit ihr den Cursor terminalunabhängig von einem Programm aus über das passende MISCINFO-File steuern können, ohne die Software dem jeweiligen Terminal anpassen zu müssen. Darüberhinaus stehen Ihnen diese Möglichkeiten zur Bildschirmsteuerung auch unter dem neuen APPLE-FORTRAN-System zur Verfügung.

Tab. 1 Beschreibung des MISCINFO-Files

Byte	**Referenz**	**Wert**	**Beschreibung**
1-62			Wird von SETUP.CODE offenbar nicht benutzt
63	4.3.4 246	0	Lead-In-Zeichen für Bildschirm
64	4.3.4 247	25	Cursor-Home-Position
65	4.3.4 246	11	Löschen bis Zeilenende
66	4.3.4 247	29	Löschen bis Bildschirmende
67	4.3.4 247	28	Cursor nach rechts
68	4.3.4 247	31	Cursor nach oben
69	4.3.4 247	8	Backspace (Linkspfeil)
70			unbekannt
71	4.3.4 247	0	Zeile löschen
72	4.3.4 246	12	Bildschirm löschen
73	4.3.4 246		Präfix-Markierungen für Bildschirm-Kontrollzeichen Bit 0 - Cursor nach oben Bit 1 - Cursor nach rechts Bit 2 - Löschen bis Zeilenende Bit 3 - Löschen bis Bildschirmende

			Bit 4 - Cursor-Home-Position
			Bit 5 - Backspace
			Bit 6 - Bildschirm löschen
			Bit 7 - Zeile löschen
74			unbekannt
75	4.3.2 243	24	Bildschirmhöhe
76			unbekannt
77	4.3.2 243	79	Bildschirmbreite
78			unbekannt
79	4.3.3 245	15	Taste zur Aufwärtsbewegung des Cursors Ctrl-O
80	4.3.3 245	12	Taste zur Abwärtsbewegung des Cursors Ctrl-L
81	4.3.3 245	8	Cursor-Links-Taste Ctrl-H
82	4.3.3 245	21	Cursor-Rechts-Taste Ctrl-U
83	4.3.3 245	3	Taste Ctrl-C für Dateiende
84	4.3.3 244	6	Tastatursperre (Flush) Ctrl-F
85	4.3.3 244	0	Break-Taste
86	4.3.3 244	19	Stop-Taste Ctrl-S
87	4.3.3 245	8	Taste zum Löschen von Zeichen Ctrl-H
88	4.3.2 243	63	Nicht ausgebbares Zeichen?
89	4.3.3 245	24	Zeichen zum Löschen einer Zeile Ctrl-X
90	4.3.3 246	27	Editor-Escape-Taste ESC
91	4.3.3 244	0	Lead-In-Zeichen für Tastatur
92	4.3.3 246	3	Annahmetaste für Editor Ctrl-C
93-512			Wird anscheinend nicht benutzt

```
(*$s+*)
unit crtutilites;

interface
   var
      LUNMISCINFO   file;

   procedure CURSORHOME;
   procedure CURSORUP;
   procedure CURSORDOWN;
   procedure CURSORLEFT;
   procedure CURSORRIGHT;
   procedure CLEARSCREEN;
   procedure CLEARLINE(LINENUMBER : integer);
   procedure ERASEOL;
   procedure ERASEOS(LINENUMBER : integer);
   procedure DISPLAY(VTAB,HTAB : integer; LINEOFTEXT : string);
   procedure PROMPT(VTAB : integer; PROMPTLINE : string);

implementation
   const
      BLOCKSIZE = 512;

   type
      CRTCOMMAND = (LEADIN, UP, RIGHT, ERASEEOL, ERASEEOS, HOME,
                    LEFT, CLEARS, CLEARL, DOWN);

      INPUTBUFFER = record
                       MISCINFO : packed array[1..BLOCKSIZE] of char;
                    end;

   var
      LOWERVTAB, UPPERVTAB,
      LOWERHTAB, UPPERHTAB,
      NUMOFBLOCKS,
      PREFIXCONTROL: integer; (* Variable mit 8 Flags zu je einem Bit, *)
                              (* die bestimmt, ob vor dem Bildschirm-  *)
                              (* befehl der Lead-In Befehl stehen muss *)
                              (* oder nicht.                           *)

      CRTCONTROL   : packed array [CRTCOMMAND] of char;
      PREFIXED     : packed array [CRTCOMMAND] of boolean;
      INPUTPOINTER: ^INPUTBUFFER;
      HEAPPOINTER : ^integer;

procedure CURSORHOME;
   begin
      if PREFIXED[HOME] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[HOME],1);
   end;

procedure CURSORUP;
   begin
      if PREFIXED[UP] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[UP],1);
   end;
```

```
procedure CURSORDOWN;
   begin
      if PREFIXED[DOWN] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[DOWN],1);
   end;

procedure CURSORLEFT;
   begin
      if PREFIXED[LEFT] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[LEFT],1);
   end;

procedure CURSORRIGHT;
   begin
      if PREFIXED[RIGHT] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[RIGHT],1);
   end;


procedure CLEARSCREEN;
   begin
      if PREFIXED[CLEARS] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[CLEARS],1);
   end;

procedure ERASEOL;
   begin
      if PREFIXED[ERASEEOL] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[ERASEEOL],1);
   end;

procedure CLEARLINE;
  begin
     if LINENUMBER < LOWERVTAB then
        LINENUMBER := LOWERVTAB
     else if LINENUMBER > UPPERVTAB then
        LINENUMBER := UPPERVTAB;
     GOTOXY(0,LINENUMBER);
     ERASEOL;
  end;

procedure ERASEOS;
   begin
      if LINENUMBER < LOWERVTAB then
         LINENUMBER := LOWERVTAB
      else if LINENUMBER > UPPERVTAB then
         LINENUMBER := UPPERVTAB;
      if PREFIXED[ERASEEOS] then
         unitwrite(1,CRTCONTROL[LEADIN],1);
      unitwrite(1,CRTCONTROL[ERASEEOS],1)
   end;
```

```
procedure DISPLAY;
   begin
      if VTAB < LOWERVTAB then
         VTAB := LOWERVTAB
      else if VTAB > UPPERVTAB then
         VTAB := UPPERVTAB;
      if HTAB < LOWERHTAB then
         HTAB := LOWERHTAB
      else if HTAB > UPPERHTAB then
         HTAB := UPPERHTAB;
      GOTOXY(HTAB,VTAB);
      WRITE(OUTPUT,LINEOFTEXT);
   end;

procedure PROMPT;
   begin
      if VTAB < LOWERVTAB then
         VTAB := LOWERVTAB
      else if VTAB > UPPERVTAB then
         VTAB := UPPERVTAB;
      GOTOXY(0,VTAB);
      WRITE(OUTPUT,PROMPTLINE);
      ERASEOL;
   end;

begin (* Unitinitialisierung *)
   RESET(LUNMISCINFO,'*SYSTEM.MISCINFO');
   MARK(HEAPPOINTER);
   NEW(INPUTPOINTER);
   with INPUTPOINTER^ do
      begin
         NUMOFBLOCKS := BLOCKREAD(LUNMISCINFO,MISCINFO,1);
         CLOSE(LUNMISCINFO);

         PREFIXCONTROL := ORD(MISCINFO[73]);

         CRTCONTROL[LEADIN   ] := MISCINFO[63];
         CRTCONTROL[UP       ] := MISCINFO[68];
         CRTCONTROL[RIGHT    ] := MISCINFO[67];
         CRTCONTROL[ERASEEOL ] := MISCINFO[66];
         CRTCONTROL[ERASEEOS ] := MISCINFO[65];
         CRTCONTROL[HOME     ] := MISCINFO[64];
         CRTCONTROL[LEFT     ] := MISCINFO[69];
         CRTCONTROL[CLEARS   ] := MISCINFO[72];
         CRTCONTROL[CLEARL   ] := MISCINFO[71];
         CRTCONTROL[DOWN     ] := CHR(10);

         PREFIXED[LEADIN   ] := false;
         PREFIXED[UP       ] := ODD(PREFIXCONTROL        );
         PREFIXED[RIGHT    ] := ODD(PREFIXCONTROL div   2);
         PREFIXED[ERASEEOL ] := ODD(PREFIXCONTROL div   4);
         PREFIXED[ERASEEOS ] := ODD(PREFIXCONTROL div   8);
         PREFIXED[HOME     ] := ODD(PREFIXCONTROL div  16);
         PREFIXED[LEFT     ] := ODD(PREFIXCONTROL div  32);
         PREFIXED[CLEARS   ] := ODD(PREFIXCONTROL div  64);
         PREFIXED[CLEARL   ] := ODD(PREFIXCONTROL div 128);
         PREFIXED[DOWN     ] := false;
```

```
         LOWERVTAB := 0;
         UPPERVTAB := ORD(MISCINFO[75]);
         LOWERHTAB := 0;
         UPPERHTAB := ORD(MISCINFO[77]);
      end;
   RELEASE(HEAPPOINTER);
end.
```

David Lieberman

Kleinbuchstaben, Invers- und Flash-Zeichen

In der März/April-Ausgabe von *Call-A.P.P.L.E* zeigte Ron DeGroat einen raffinierten Patch für SYSTEM.APPLE, mit dem man Kleinbuchstaben unter Pascal verwenden kann, wenn man den *Paymar Lower Case Adapter* benutzt. Ich sprach mit ihm über diesen Patch und brachte den Einwand vor, daß damit die SYSTEM.APPLE-Datei permanent geändert würde. Er erwähnt diesen Nachteil auch in seinem Artikel und rät jedem (jeder) Anwender(in), sich eine spezielle Diskette mit diesem Patch anzufertigen.

Nachdem ich mit Ron gesprochen hatte, kam ich auf die Idee, diesen Patch in Maschinensprache zu implementieren und die SYSTEM.APPLE-Modifikation direkt in der Ram-Card vorzunehmen. Damit ist es nicht mehr notwendig, die Diskettenversion dieses Datei zu ändern. Der Patch läßt sich daher durch einen einfachen Kaltstart wieder löschen. Wenn Sie den Patch bei jedem Start Ihres Pascal-Systems anwenden wollen, können Sie die LCA-Prozedur in Ihre SYSTEM-STARTUP-Datei einbauen.

In Abb.1 finden Sie ein Pascal-Programmbeispiel, in dem die LCA-Prozedur aufgerufen wird und damit Kleinschreibung ermöglicht. In Abb.2 finden Sie die Assemblerprozeduren, die den Patch ausführen. Beachten Sie bitte: Die Darstellung von "Pseudo-Großbuchstaben" wird verhindert: Wenn der Paymar-Patch benutzt wird, lassen sich Großbuchstaben nach Eingabe des Ctrl-R Befehls nicht mehr invertiert darstellen.

Wenn Sie invertierte oder blinkende Zeichen darstellen wollen, können Sie das jedoch mit den in Abb.2 gezeigten Prozeduren erreichen. Für negative Textausgabe können Sie beispielsweise folgende Befehlssequenz verwenden:

```
Gotoxy (0,4);
Inverse;
Write (''DIESER TEXT IST INVERTIERT'');
Normal;
```

```
Program LcaPatch;

Procedure Lca;  External;

(* Lca muss in LcaPatch gelinkt werden *)

Begin
  Lca;
  Gotoxy(8,8);
  Write('Pascal 1.1 mit Kleinbuchstabenanzeige');
End.
```

Abb. 1 Programmbeispiel für Lca-Aufruf

```
        .PROC LCA

;DIESE PROZEDUR AENDERT DAS PASCAL 1.1 SO IM BIOS,
;DASS KLEINBUCHSTABEN MIT EINEM LOWERCASE-ADAPTER
;AUSGEGEBEN WERDEN KOENNEN. DIESER PATCH FUNKTIONIERT
;NUR MIT PASCAL 1.1

;VON DAVE LIEBERMAN 12.6.81

ADDR1   .EQU ODAAB
RAMON   .EQU OC083
RAMCLR  .EQU OC088

        LDA RAMON       ;ZWEITE 4-K-BANK ANWAEHLEN
        LDA RAMON       ;SCHREIBZUGRIFF ERMOEGLICHEN

        LDA #176.       ;GROSSBUCHSTABENKONVERSION UNTERDRUECKEN
        STA ADDR1
        LDA #02.
        STA ADDR1+1
        LDA #0          ;PSEUDOGROSSBUCHSTABEN ABSCHALTEN
        STA ADDR1+239.

        LDA RAMCLR      ;ZURUECKSCHALTEN AUF ERSTE BANK
        RTS

        .END

.PROC   INVERSE          ;KEINE PARAMETER

;PROCEDURE INVERSE;
;------------------
;
;DIE NAECHSTEN DREI SUBROUTINEN ERMOEGLICHEN ES,
;UNTER PASCAL INVERSE, NORMALE ODER BLINKENDE
;ZEICHEN IM NORMALEN APPLE TEXTFENSTER AUSZUGEBEN.
;BEIM INVERSE-MODUS ERSCHEINEN DIE ZEICHEN SCHWARZ
;AUF HELLEM HINTERGRUND.
;
```

```
        LDA     0C083   ;ZWEITE 4-K-BANK AKTIVIEREN
        LDA     0C083   ;SCHREIBZUGRIFF ERMOEGLICHEN
        LDA     #00
        STA     0DAB0   ;BITS 6 & 7 LOESCHEN
        LDA     0C088   ;ERSTE BANK EINSCHALTEN
                        ;UND GEGEN UEBERSCHREIBEN SCHUETZEN
        RTS

    .PROC   NORMAL          ;KEINE PARAMETER
;
;PROCEDURE NORMAL;
;----------------
;
;NORMAL GIBT ZEICHEN WEISS AUF
;SCHWARZEM HINTERGRUND AUS
;
    LDA     0C083
    LDA     0C083
    LDA     #80
    STA     0DAB0   ;BIT 7 SETZEN
    LDA     0C088
    RTS

    .PROC   FLASH           ;KEINE PARAMETER
;
;PROCEDURE FLASH
;----------------
;
;FLASH GIBT BLINKENDE ZEICHEN AUS,
;ABWECHSELND NORMAL- UND INVERSE-MODUS
;
    LDA     0C083
    LDA     0C083
    LDA     #40
    STA     0DAB0   ;BIT 6 SETZEN
    LDA     0C088
    RTS
```

Abb. 2 Prozeduren für Invers- und Flash-Darstellung

Die einzige Einschränkung ist dabei, daß sich Kleinbuchstaben nicht invertiert oder blinkend ausgeben lassen. Bei dem Versuch, Kleinbuchstaben zu invertieren oder blinken zu lassen, erhalten Sie nur Unsinn auf dem Bildschirm.

Das Programm "LcaPatch" zeigt, wie man echte Kleinbuchstaben mit einem "Lower Case Adapter" ausgeben kann. Achten Sie darauf, daß dieser Patch nur mit der Pascal 1.1-Version funktioniert und mit anderen Modifikationen des BIOS (BASIC Input Output System) möglicherweise nicht kompatibel ist.

Wenn man "LcaPatch" korrekt kompiliert, eingebunden und in SYSTEM.STARTUP umbenannt hat, wird das Programm bei jedem Booten des Betriebssystems aufgerufen. Da keine permanenten Änderungen vorgenommen werden, ist dieser Patch relativ sicher. Will man die Anzeige von Kleinbuchstaben abschalten, braucht man nur den Namen von SYSTEM.STARTUP zu ändern und das Betriebssystem neu zu booten.

Chris Wilson

P-CODE DECODER

```
(*$C COPYRIGHT 1981 CHRIS WILSON *)

(* BENUTZEN SIE ZUM EDITIEREN UND KOMPILIEREN DEN SYSTEM-SWAPMODUS *)

PROGRAM DECODE;

TYPE
  WORD = PACKED RECORD
    CASE INTEGER OF
    0: (B: PACKED ARRAY [0..1] OF 0..255);
    1: (C: PACKED ARRAY [0..1] OF CHAR);
    2: (H: PACKED ARRAY [0..3] OF 0..15);
    3: (I: INTEGER);
    4: (P: ^ WORD)
    END;
  MTYPES = (UNDEF, PCODEMOST, PCODELEAST, PDP11, M8080,
    Z80, GA440, M6502, M6800, TI9900);
  SDRECORD = RECORD
    DISKINFO: ARRAY [0..15] OF
      RECORD
      CODELENG,
      CODEADDR: INTEGER
      END;
    SEGNAME: ARRAY [0..15] OF
      PACKED ARRAY [0..7] OF CHAR;
    SEGKIND: ARRAY [0..15] OF
      (LINKED, HOSTSEG, SEGPROC, UNITSEG,
       SEPRTSEG, UNLINKEDINTRINS,
       LINKEDINTRINS, DATASEG);
    TEXTADDR: ARRAY[0..15] OF INTEGER;
    SEGINFO: PACKED ARRAY [0..15] OF
      PACKED RECORD
      SEGNUM: 0..255;
      MTYPE: MTYPES;
      UNUSED: 0..1;
      VERSION: 0..7
      END;
    (* UND ANDERE GUTE SACHEN *)
    END;
  FREEUNION = RECORD
    CASE INTEGER OF
    1: (BUF: PACKED ARRAY [0..511] OF 0..255);
```

```
    2: (DICT: SDRECORD);
    END;
  STRING1 = PACKED ARRAY [0..1] OF CHAR;
  STRING3 = PACKED ARRAY [0..3] OF CHAR;
  STRING7 = STRING[7];
  PTYPE = (UB,SB,DB,B,W,X0,X1,X2,X3,X4,X5,X6,X7,XX);
  OPREC = RECORD
    MNEMONIC: STRING7;
    P1,
    P2: PTYPE;
    END;

VAR
  PDCOUNT,
  FIRSTADDR,
  FIRSTBLOCK,
  CURRENTBLOCK: INTEGER;
  ADDR: STRING3;
  F: TEXT;
  SOURCEFILE: FILE;
  SOURCENAME,
  DESTNAME: STRING;
  OPCODE: ARRAY [0..255] OF OPREC;
  BUF: PACKED ARRAY [0..511] OF 0..255;
  SD: FREEUNION;
  PD: ARRAY [0..149] OF INTEGER;
  HEXDIGIT: PACKED ARRAY [0..15] OF CHAR;

PROCEDURE WAIT;
        (*====*)

BEGIN
IF DESTNAME = 'CONSOLE:' THEN
  BEGIN
  WRITELN;
  WRITE('[WEITER MIT RETURN]');
  READLN;
  END;
END;

PROCEDURE SKIP;
        (*====*)

BEGIN
IF DESTNAME = 'CONSOLE:' THEN
  PAGE(F)
ELSE
  WRITELN(F);
END;

PROCEDURE READBLOCK(LOC: INTEGER);
        (*=========*)

BEGIN
IF LOC < FIRSTADDR THEN
  REPEAT
    CURRENTBLOCK := PRED(CURRENTBLOCK);
    FIRSTADDR := FIRSTADDR-512;
  UNTIL LOC >= FIRSTADDR
```

```
ELSE IF LOC >= (FIRSTADDR+512) THEN
  REPEAT
    CURRENTBLOCK := SUCC(CURRENTBLOCK);
    FIRSTADDR := FIRSTADDR+512;
  UNTIL LOC < (FIRSTADDR+512);
IF BLOCKREAD(SOURCEFILE,BUF,1,CURRENTBLOCK) <> 1 THEN
  BEGIN
  WRITELN('BLOCKREAD: FEHLER BEIM LESEN DES QUELLFILES');
  EXIT(DECODE);
  END;
END;

FUNCTION BYTEVAL(LOC: INTEGER): INTEGER;
       (*======*)

BEGIN
IF (LOC >= FIRSTADDR) AND (LOC < FIRSTADDR+512) THEN
  BYTEVAL := BUF[LOC-FIRSTADDR]
ELSE
  BEGIN
  READBLOCK(LOC);
  BYTEVAL := BYTEVAL(LOC);
  END;
END;

FUNCTION WORDVAL(LOC: INTEGER): INTEGER;
       (*======*)

VAR
  W: WORD;

BEGIN
W.B[0] := BYTEVAL(LOC);
W.B[1] := BYTEVAL(SUCC(LOC));
WORDVAL := W.I;
END;

PROCEDURE HEXBYTE(VALUE: INTEGER; VAR HEX: STRING1);
        (*======*)

VAR
  W: WORD;

BEGIN
W.I := VALUE;
HEX[0] := HEXDIGIT[W.H[1]];
HEX[1] := HEXDIGIT[W.H[0]];
END;

PROCEDURE HEXWORD(VALUE: INTEGER; VAR HEX: STRING3);
        (*======*)

VAR
  W: WORD;

BEGIN
W.I := VALUE;
HEX[0] := HEXDIGIT[W.H[3]];
HEX[1] := HEXDIGIT[W.H[2]];
```

```
HEX[2] := HEXDIGIT[W.H[1]];
HEX[3] := HEXDIGIT[W.H[0]];
END;

PROCEDURE DECODEPROC(PROC: INTEGER);
        (*==========*)

VAR
  IPC,
  JTAB,
  LEXLEVEL,
  ENTERIC,
  EXITIC,
  PARAMSIZE,
  DATASIZE,
  LASTCODE: INTEGER;
  HEX: STRING3;


PROCEDURE ONEOP;
        (*=====*)

VAR
  I,
  MIN,
  MAX: INTEGER;
  BYTE: STRING1;
  HEX: STRING3;

PROCEDURE HANDLEDB;
  BEGIN
  IPC := SUCC(IPC);
  HEXBYTE(BYTEVAL(IPC),BYTE);
  WRITE(F, BYTE:3);
  END;

PROCEDURE HANDLEB;
  BEGIN
  IPC := SUCC(IPC);
  IF BYTEVAL(IPC) > 127 THEN
    BEGIN
    HEXBYTE(BYTEVAL(IPC)-128,BYTE);
    WRITE(F, BYTE:3);
    IPC := SUCC(IPC);
    HEXBYTE(BYTEVAL(IPC),BYTE);
    WRITE(F, BYTE);
    END
  ELSE
    BEGIN
    HEXBYTE(BYTEVAL(IPC),BYTE);
    WRITE(F, BYTE:3);
    END;
  END;

PROCEDURE HANDLEW;
  BEGIN
  HEXBYTE(BYTEVAL(IPC+2),BYTE);
  WRITE(F, BYTE:3);
  HEXBYTE(BYTEVAL(IPC+1),BYTE);
  WRITE(F, BYTE);
```

```
    IPC := IPC+2;
    END;

  PROCEDURE HANDLECSP;
    VAR S: STRING;
    BEGIN
    S := '';
    CASE BYTEVAL(IPC) OF
     0: S := ' (IOCHECK)';
     1: S := ' (NEW)';
     2: S := ' (MOVELEFT)';
     3: S := ' (MOVERIGHT)';
     4: S := ' (EXIT)';
     5: S := ' (UNITREAD)';
     6: S := ' (UNITWRITE)';
     7: S := ' (IDSEARCH)';
     8: S := ' (TREESEARCH)';
     9: S := ' (TIME)';
    10: S := ' (FILLCHAR)';
    11: S := ' (SCAN)';
    21: S := ' (LOAD RESIDENT SEGMENT)';
    22: S := ' (UNLOAD RESIDENT SEGMENT)';
    23: S := ' (TRUNC)';
    24: S := ' (ROUND)';
    32: S := ' (MARK)';
    33: S := ' (RELEASE)';
    34: S := ' (IORESULT)';
    35: S := ' (UNITBUSY)';
    36: S := ' (PWROFTEN)';
    37: S := ' (UNITWAIT)';
    38: S := ' (UNITCLEAR)';
    39: S := ' (HALT)';
    40: S := ' (MEMAVAIL)';
    END; (* CASE *)

  WRITE(F, S);
  END;

PROCEDURE HANDLECXP;
  VAR S: STRING;
  BEGIN
  HANDLEDB;
  IF BYTEVAL(PRED(IPC)) = 0 THEN
    BEGIN
    S := '';
    CASE BYTEVAL(IPC) OF
     2: S := ' (EXECERROR)';
     3: S := ' (BUILD FIB)';
     5: S := ' (RESET/REWRITE)';
     6: S := ' (CLOSE)';
     7: S := ' (GET)';
     8: S := ' (PUT)';
    10: S := ' (EOF)';
    11: S := ' (EOLN)';
    12: S := ' (READ INTEGER)';
    13: S := ' (WRITE INTEGER)';
    16: S := ' (READ CHAR)';
    17: S := ' (WRITE CHAR)';
    18: S := ' (READ STRING)';
    19: S := ' (WRITE STRING)';
```

```
    20: S := ' (WRITE ARRAY OF CHAR)';
    21: S := ' (READLN)';
    22: S := ' (WRITELN)';
    23: S := ' (CONCAT)';
    24: S := ' (INSERT)';
    25: S := ' (COPY)';
    26: S := ' (DELETE)';
    27: S := ' (POS)';
    28: S := ' (BLOCK READ/WRITE)';
    29: S := ' (GOTOXY)';
    END; (* CASE *)
    WRITE(F, S);
    END;
  END;

BEGIN (* ONEOP *)
WITH OPCODE[BYTEVAL(IPC)] DO
  BEGIN
  HEXWORD(IPC,HEX);
  HEXBYTE(BYTEVAL(IPC),BYTE);
  WRITE(F, HEX, ' ', MNEMONIC, ' (', BYTE, ')', ' ':7-LENGTH(MNEMONIC));
  CASE P1 OF
  UB, SB, DB:
    HANDLEDB;
  B:
    HANDLEB;
  W:
    HANDLEW;
  XX:
    BEGIN
    END;
  END; (* CASE *)
  CASE P2 OF
  UB, SB, DB:
    HANDLEDB;
  B:
    HANDLEB;
  W:
    HANDLEW;
  X0:
    HANDLECSP;
  X1:
    BEGIN
    (* LSA, LSP *)
    WRITE(F, ' ''');
    FOR I := 1 TO BYTEVAL(IPC) DO
      BEGIN
      IPC := SUCC(IPC);
      IF I MOD 16 = 0 THEN
        BEGIN
        WRITELN(F, '''');
        WRITE(F, ' ':21, '''');
        END;
      IF BYTEVAL(IPC) > 31 THEN
        WRITE(F, CHR(BYTEVAL(IPC)))
      ELSE
        WRITE(F, '.');
      END;
    WRITE(F, '''');
    END;
```

```
X2:
  BEGIN
  (* XJP *)
  IF NOT ODD(IPC) THEN
    (* MUSS WORTLAENGE HABEN *)
    IPC := SUCC(IPC);
  HANDLEW;
  HANDLEW;
  HANDLEW;
  MIN := WORDVAL(IPC-5);
  MAX := WORDVAL(IPC-3);
  FOR I := MIN TO MAX DO
    BEGIN
    WRITELN(F);
    WRITE(F, ' ':19);
    HANDLEW;
    HEXWORD(PRED(IPC)-WORDVAL(PRED(IPC)),HEX);
    WRITE(F, ' (', HEX, ')');
    END;
  END;
X3:
  BEGIN
  (* EQU, ETC. *)
  CASE BYTEVAL(IPC) OF
   2: WRITE(F, ' (REAL)');
   4: WRITE(F, ' (STRING)');
   6: WRITE(F, ' (BOOLEAN)');
   8: WRITE(F, ' (SET)');
  10: BEGIN
      HANDLEB;
      WRITE(F, ' (BYTE ARRAY)');
      END;
  12: BEGIN
      HANDLEB;
      WRITE(F, ' (WORD)');
      END;
  END; (* CASE *)
  END;
X4:
  BEGIN
  (* LDC *)
  MAX := BYTEVAL(IPC);
  IF NOT ODD(IPC) THEN
    (* MUSS WORTLAENGE HABEN *)
    IPC := SUCC(IPC);
  FOR I := 1 TO MAX DO
    BEGIN
    WRITELN(F);
    WRITE(F, ' ':19);
    HANDLEW;
    END;
  END;
X5:
  HANDLECXP;
X6:
  BEGIN
  (* FJP, UJP, EFJ, NFJ *)
  I := BYTEVAL(IPC);  (* SPRUNGOFFSET *)
  IF I < 128 THEN
```

```
    HEXWORD(SUCC(IPC)+I,HEX)
  ELSE
    BEGIN
    I := JTAB-(256-I);
    HEXWORD(I-WORDVAL(I),HEX);
    END;
  WRITE(F, ' (', HEX, ')');
  END;
X7:
  BEGIN
  (* RNP, RBP *)
  IF IPC >= EXITIC THEN
    LASTCODE := IPC;
  END;
  XX:
    BEGIN
    END;
  END; (* CASE *)
  END; (* WITH *)
WRITELN(F);
END;

BEGIN (* DECODEPROC *)
JTAB := PD[PROC];
IF JTAB < 0 THEN
  BEGIN
  WRITELN;
  WRITELN('>>> FALSCHE PROZEDURADRESSE <<<');
  END
ELSE
  BEGIN
  LEXLEVEL := BYTEVAL(SUCC(JTAB));
  IF LEXLEVEL > 127 THEN
    LEXLEVEL := LEXLEVEL-256;
  ENTERIC := (JTAB-2)-WORDVAL(JTAB-2);
  EXITIC := (JTAB-4)-WORDVAL(JTAB-4);
  PARAMSIZE := WORDVAL(JTAB-6);
  DATASIZE := WORDVAL(JTAB-8);
  LASTCODE := JTAB-9;
  SKIP;
  WRITELN(F, 'PROCEDURCODE:');
  WRITELN(F, '-------------');
  WRITELN(F);
  WRITELN(F, 'LEX LEVEL ', LEXLEVEL, ', PROZEDUR ', BYTEVAL(JTAB));
  HEXWORD(ENTERIC,HEX);
  WRITELN(F, 'ENTER IC ', ENTERIC, ' (', HEX, ')');
  HEXWORD(EXITIC,HEX);
  WRITELN(F, 'EXIT IC ', EXITIC, ' (', HEX, ')');
  HEXWORD(PARAMSIZE,HEX);
  WRITELN(F, 'PARAMETERGROESSE ', PARAMSIZE, ' (', HEX, ')');
  HEXWORD(DATASIZE,HEX);
  WRITELN(F, 'DATENGROESSE ', DATASIZE, ' (', HEX, ')');
  WRITELN(F);
  IPC := ENTERIC;
  IF LEXLEVEL < -1 THEN
    WRITELN('>>> FALSCHER LEXIKALISCHER LEVEL <<<')
  ELSE IF ENTERIC < 0 THEN
    WRITELN('>>> ENTER IC FALSCH <<<')
  ELSE IF EXITIC < 0 THEN
```

```
    WRITELN('>>> EXIT IC FALSCH <<<')
  ELSE
    REPEAT
      ONEOP;
      IPC := SUCC(IPC);
    UNTIL IPC > LASTCODE;
  END;
WAIT;
END;

PROCEDURE CHOOSEPROC;
        (*=========*)

VAR
  I: INTEGER;
  DONE: BOOLEAN;

BEGIN
REPEAT
  PAGE(OUTPUT);
  WRITELN('ZU DECODIERENDE PROZEDUR:');
  WRITELN;
  WRITELN('[1..', PDCOUNT, ']');
  WRITELN;
  WRITELN;
  WRITELN('-1 FUER ENDE');
  WRITELN;
  WRITE('PROZEDUR: ');
  READLN(I);
  DONE := I < 0;
  IF I IN [1..PDCOUNT] THEN
    DECODEPROC(I);
UNTIL DONE;
END;

PROCEDURE READPROCDICT(SEG: INTEGER);
        (*==========*)

VAR
  I,
  LOC,
  SEGLENGTH: INTEGER;
  HEX: STRING3;

BEGIN
WITH SD.DICT DO
  BEGIN
  FIRSTADDR := 0;
  FIRSTBLOCK := DISKINFO[SEG].CODEADDR;
  CURRENTBLOCK := FIRSTBLOCK;
  SEGLENGTH := DISKINFO[SEG].CODELENG;
  END;
LOC := PRED(SEGLENGTH);
READBLOCK(LOC);
PDCOUNT := BYTEVAL(LOC);
LOC := PRED(LOC);
SKIP;
WRITELN(F, 'PROZEDURDICTIONARY:');
WRITELN(F, '-------------------');
WRITELN(F);
WRITELN(F, 'SEGMENT ', BYTEVAL(LOC));
```

```
WRITELN(F, 'PROZEDURZAEHLER ', PDCOUNT);
WRITELN(F);
FOR I := 1 TO PDCOUNT DO
  BEGIN
  LOC := LOC-2;
  PD[I] := LOC-WORDVAL(LOC);
  HEXWORD(PD[I],HEX);
  WRITELN(F, 'PROZEDUR ', I:2, ', ADRESSE ', PD[I]:5, ' (', HEX, ')');
  END;
WAIT;
CHOOSEPROC;
END;


PROCEDURE CHOOSESEGMENT;
        (*=============*)

VAR
  I: INTEGER;
  ANSWER: CHAR;
  DONE: BOOLEAN;

BEGIN
WITH SD.DICT DO
  REPEAT
    PAGE(OUTPUT);
    WRITELN('ZU ANALYSIERENDES SEGMENT:');
    WRITELN;
    FOR I := 0 TO 15 DO
      IF SEGNAME[I] <> '        ' THEN
        WRITELN(I:2, ' ', SEGNAME[I]);
    WRITELN;
    WRITELN('-1 FUER ENDE');
    WRITELN;
    WRITE('SEGMENT: ');
    READLN(I);
    DONE := I < 0;
    IF I IN [0..15] THEN
      IF SEGINFO[I].MTYPE <> PCODELEAST THEN
        BEGIN
        WRITELN('SEGMENT KEIN P-CODE ');
        IF SEGINFO[I].MTYPE IN [UNDEF,PCODEMOST] THEN
          BEGIN
          WRITE('TROTZDEM VERSUCHEN ZU DECODIEREN (J/N): ');
          READLN(ANSWER);
          IF ANSWER IN ['J','j'] THEN
            READPROCDICT(I);
          END
        ELSE
          WAIT;
        END
      ELSE
        READPROCDICT(I);
  UNTIL DONE;
END;
```

```
PROCEDURE READSEGDICT;
        (*==========*)

VAR
  I: INTEGER;
  S: STRING;

BEGIN
IF BLOCKREAD(SOURCEFILE,SD.BUF,1,0) <> 1 THEN
  BEGIN
  WRITELN('FEHLER BEIM LESEN DES SEGMENT-DICTIONARIES');
  EXIT(DECODE);
  END;
WITH SD.DICT DO
  BEGIN
  I := 0;
  SKIP;
  WRITELN(F, 'SEGMENT DICTIONARY:');
  WRITELN(F, '------------------');
  REPEAT
    IF SEGNAME[I] <> '        ' THEN
      WITH SEGINFO[I] DO
        BEGIN
        WRITELN(F);
        WRITELN(F, 'SEGMENT #', SEGNUM);
        WITH DISKINFO[I] DO
          WRITELN(F, 'LAENGE ', CODELENG, ', ADRESSE ', CODEADDR);
        WRITELN(F, 'SYSTEM VERSION = ', VERSION);
        S := '';
        CASE MTYPE OF
        UNDEF:
          S := 'UNDEFINIERT';
        PCODEMOST:
          S := 'P-CODE (MSB ZUERST)';
        PCODELEAST:
          S := 'P-CODE (LSB ZUERST)';
        PDP11:
          S := 'PDP11';
        M8080:
          S := '8080';
        Z80:
          S := 'Z80';
        GA440:
          S := 'GA440';
        M6502:
          S := '6502';
        M6800:
          S := '6800';
        TI9900:
          S := 'TI9900';
        END; (* CASE *)
        WRITELN(F, 'CODETYP IST ', S);
        S := '';
        CASE SEGKIND[I] OF
        LINKED:
          S := 'LINKED';
        HOSTSEG:
          S := 'HOST SEGMENT';
        SEGPROC:
          S := 'SEGMENTPROZEDUR';
```

```
        UNITSEG:
          S := 'UNITSEGMENT';
        SEPRTSEG:
          S := 'SEPARATE SEGMENT';
        UNLINKEDINTRINS:
          S := 'UNLINKED INTRINSIC';
        LINKEDINTRINS:
          S := 'LINKED INTRINSIC';
        DATASEG:
          S := 'DATA SEGMENT';
        END; (* CASE *)
        WRITELN(F, SEGNAME[I], ' (', S, ')');
        END;
    I := SUCC(I);
  UNTIL I > 15;
  END;
WAIT;
CHOOSESEGMENT;
END;

PROCEDURE INITIALIZE;
        (*==========*)

VAR
  I: INTEGER;
  S: STRING7;

PROCEDURE INIT(OP: INTEGER; MNE:STRING7; X1, X2: PTYPE);
  BEGIN
  WITH OPCODE[OP] DO
    BEGIN
    MNEMONIC := MNE;
    P1 := X1;
    P2 := X2;
    END;
  END;

PROCEDURE INIT1;
  BEGIN
  INIT(128,'ABI'   ,XX,XX);  (* 80 *)
  INIT(129,'ABR'   ,XX,XX);  (* 81 *)
  INIT(130,'ADI'   ,XX,XX);  (* 82 *)
  INIT(131,'ADR'   ,XX,XX);  (* 83 *)
  INIT(132,'LAND'  ,XX,XX);  (* 84 *)
  INIT(133,'DIF'   ,XX,XX);  (* 85 *)
  INIT(134,'DVI'   ,XX,XX);  (* 86 *)
  INIT(135,'DVR'   ,XX,XX);  (* 87 *)
  INIT(136,'CHK'   ,XX,XX);  (* 88 *)
  INIT(137,'FLO'   ,XX,XX);  (* 89 *)
  INIT(138,'FLT'   ,XX,XX);  (* 8A *)
  INIT(139,'INN'   ,XX,XX);  (* 8B *)
  INIT(140,'INT'   ,XX,XX);  (* 8C *)
  INIT(141,'LOR'   ,XX,XX);  (* 8D *)
  INIT(142,'MODI'  ,XX,XX);  (* 8E *)
  INIT(143,'MPI'   ,XX,XX);  (* 8F *)
  INIT(144,'MPR'   ,XX,XX);  (* 90 *)
  INIT(145,'NGI'   ,XX,XX);  (* 91 *)
  INIT(146,'NGR'   ,XX,XX);  (* 92 *)
  INIT(147,'LNOT'  ,XX,XX);  (* 93 *)
  INIT(148,'SRS'   ,XX,XX);  (* 94 *)
```

```
  INIT(149,'SBI'   ,XX,XX);  (* 95 *)
  INIT(150,'SBR'   ,XX,XX);  (* 96 *)
  INIT(151,'SGS'   ,XX,XX);  (* 97 *)
  INIT(152,'SQI'   ,XX,XX);  (* 98 *)
  INIT(153,'SQR'   ,XX,XX);  (* 99 *)
  INIT(154,'STO'   ,XX,XX);  (* 9A *)
  INIT(155,'IXS'   ,XX,XX);  (* 9B *)
  INIT(156,'UNI'   ,XX,XX);  (* 9C *)
  INIT(157,'LDE'   ,UB, B);  (* 9D *)
  INIT(158,'CSP'   ,UB,X0);  (* 9E *)
  INIT(159,'LDCN'  ,XX,XX);  (* 9F *)
  INIT(160,'ADJ'   ,UB,XX);  (* A0 *)
  INIT(161,'FJP'   ,SB,X6);  (* A1 *)
  INIT(162,'INC'   , B,XX);  (* A2 *)
  INIT(163,'IND'   , B,XX);  (* A3 *)
  INIT(164,'IXA'   , B,XX);  (* A4 *)
  INIT(165,'LAO'   , B,XX);  (* A5 *)
  INIT(166,'LSA'   ,UB,X1);  (* A6 *)
  INIT(167,'LAE'   ,UB, B);  (* A7 *)
  INIT(168,'MOV'   , B,XX);  (* A8 *)
  INIT(169,'LDO'   , B,XX);  (* A9 *)
  INIT(170,'SAS'   ,UB,XX);  (* AA *)
  INIT(171,'SRO'   , B,XX);  (* AB *)
  INIT(172,'XJP'   ,XX,X2);  (* AC *)
  INIT(173,'RNP'   ,DB,X7);  (* AD *)
  INIT(174,'CIP'   ,UB,XX);  (* AE *)
  INIT(175,'EQU'   ,DB,X3);  (* AF *)
  END;

PROCEDURE INIT2;
  BEGIN
  INIT(176,'GEQ'   ,DB,X3);  (* B0 *)
  INIT(177,'GRT'   ,DB,X3);  (* B1 *)
  INIT(178,'LDA'   ,DB, B);  (* B2 *)
  INIT(179,'LDC'   ,UB,X4);  (* B3 *)
  INIT(180,'LEQ'   ,DB,X3);  (* B4 *)
  INIT(181,'LES'   ,DB,X3);  (* B5 *)
  INIT(182,'LOD'   ,DB, B);  (* B6 *)
  INIT(183,'NEQ'   ,DB,X3);  (* B7 *)
  INIT(184,'STR'   ,DB, B);  (* B8 *)
  INIT(185,'UJP'   ,SB,X6);  (* B9 *)
  INIT(186,'LDP'   ,XX,XX);  (* BA *)
  INIT(187,'STP'   ,XX,XX);  (* BB *)
  INIT(188,'LDM'   ,UB,XX);  (* BC *)
  INIT(189,'STM'   ,UB,XX);  (* BD *)
  INIT(190,'LDB'   ,XX,XX);  (* BE *)
  INIT(191,'STB'   ,XX,XX);  (* BF *)
  INIT(192,'IXP'   ,UB,UB);  (* C0 *)
  INIT(193,'RBP'   ,DB,X7);  (* C1 *)
  INIT(194,'CBP'   ,UB,XX);  (* C2 *)
  INIT(195,'EQUI'  ,XX,XX);  (* C3 *)
  INIT(196,'GEQI'  ,XX,XX);  (* C4 *)
  INIT(197,'GRTI'  ,XX,XX);  (* C5 *)
  INIT(198,'LLA'   , B,XX);  (* C6 *)
  INIT(199,'LDCI'  , W,XX);  (* C7 *)
  INIT(200,'LEQI'  ,XX,XX);  (* C8 *)
  INIT(201,'LESI'  ,XX,XX);  (* C9 *)
  INIT(202,'LDL'   , B,XX);  (* CA *)
  INIT(203,'NEQI'  ,XX,XX);  (* CB *)
  INIT(204,'STL'   , B,XX);  (* CC *)
```

```
  INIT(205,'CXP'   ,UB,X5);  (* CD *)
  INIT(206,'CLP'   ,UB,XX);  (* CE *)
  INIT(207,'CGP'   ,UB,XX);  (* CF *)
  INIT(208,'LPA'   ,UB,X1);  (* D0 *)
  INIT(209,'STE'   ,UB, B);  (* D1 *)
  INIT(210,'NOP'   ,XX,XX);  (* D2 *)
  INIT(211,'EFJ'   ,SB,X6);  (* D3 *)
  INIT(212,'NFJ'   ,SB,X6);  (* D4 *)
  INIT(213,'BPT'   , B,XX);  (* D5 *)
  INIT(214,'XIT'   ,XX,XX);  (* D6 *)
  INIT(215,'NOP'   ,XX,XX);  (* D7 *)
  END;

PROCEDURE INIT3;
  BEGIN
  INIT(216,'SLDL1' ,XX,XX);  (* D8 *)
  INIT(217,'SLDL2' ,XX,XX);  (* D9 *)
  INIT(218,'SLDL3' ,XX,XX);  (* DA *)
  INIT(219,'SLDL4' ,XX,XX);  (* DB *)
  INIT(220,'SLDL5' ,XX,XX);  (* DC *)
  INIT(221,'SLDL6' ,XX,XX);  (* DD *)
  INIT(222,'SLDL7' ,XX,XX);  (* DE *)
  INIT(223,'SLDL8' ,XX,XX);  (* DF *)
  INIT(224,'SLDL9' ,XX,XX);  (* E0 *)
  INIT(225,'SLDL10',XX,XX);  (* E1 *)
  INIT(226,'SLDL11',XX,XX);  (* E2 *)
  INIT(227,'SLDL12',XX,XX);  (* E3 *)
  INIT(228,'SLDL13',XX,XX);  (* E4 *)
  INIT(229,'SLDL14',XX,XX);  (* E5 *)
  INIT(230,'SLDL15',XX,XX);  (* E6 *)
  INIT(231,'SLDL16',XX,XX);  (* E7 *)
  INIT(232,'SLDO1' ,XX,XX);  (* E8 *)
  INIT(233,'SLDO2' ,XX,XX);  (* E9 *)
  INIT(234,'SLDO3' ,XX,XX);  (* EA *)
  INIT(235,'SLDO4' ,XX,XX);  (* EB *)
  INIT(236,'SLDO5' ,XX,XX);  (* EC *)
  INIT(237,'SLDO6' ,XX,XX);  (* ED *)
  INIT(238,'SLDO7' ,XX,XX);  (* EE *)
  INIT(239,'SLDO8' ,XX,XX);  (* EF *)
  INIT(240,'SLDO9' ,XX,XX);  (* F0 *)
  INIT(241,'SLDO10',XX,XX);  (* F1 *)
  INIT(242,'SLDO11',XX,XX);  (* F2 *)
  INIT(243,'SLDO12',XX,XX);  (* F3 *)
  INIT(244,'SLDO13',XX,XX);  (* F4 *)
  INIT(245,'SLDO14',XX,XX);  (* F5 *)
  INIT(246,'SLDO15',XX,XX);  (* F6 *)
  INIT(247,'SLDO16',XX,XX);  (* F7 *)
  INIT(248,'SIND0' ,XX,XX);  (* F8 *)
  INIT(249,'SIND1' ,XX,XX);  (* F9 *)
  INIT(250,'SIND2' ,XX,XX);  (* FA *)
  INIT(251,'SIND3' ,XX,XX);  (* FB *)
  INIT(252,'SIND4' ,XX,XX);  (* FC *)
  INIT(253,'SIND5' ,XX,XX);  (* FD *)
  INIT(254,'SIND6' ,XX,XX);  (* FE *)
  INIT(255,'SIND7' ,XX,XX);  (* FF *)
  END;
```

```
BEGIN (* INITIALIZE *)
FOR I := 0 TO 127 DO
  BEGIN
  STR(I,S);
  S := CONCAT('SLDC',S);
  INIT(I,S,XX,XX);
  END;
INIT1;
INIT2;
INIT3;
END;


(* ========= HAUPTPROGRAMM ========= *)

BEGIN
HEXDIGIT := '0123456789ABCDEF';
PAGE(OUTPUT);
WRITELN('DECODE (25.6.81)');
WRITELN('COPYRIGHT 1981 CHRIS WILSON');
WRITELN;
WRITELN('INITIALISIERUNG...');
INITIALIZE;
WRITELN;
WRITE('QUELLFILE: ');
READLN(SOURCENAME);
IF SOURCENAME = '' THEN
  EXIT(DECODE);
IF POS('.CODE',SOURCENAME) = 0 THEN
  IF POS('SYSTEM.',SOURCENAME) = 0 THEN
    IF SOURCENAME[LENGTH(SOURCENAME)] <> '.' THEN
      SOURCENAME := CONCAT(SOURCENAME,'.CODE')
    ELSE
      DELETE(SOURCENAME,LENGTH(SOURCENAME),1);
WRITE('ZIELFILE: ');
READLN(DESTNAME);
IF DESTNAME = '' THEN
  DESTNAME := 'CONSOLE:'
ELSE IF POS('.TEXT',DESTNAME) = 0 THEN
  IF DESTNAME[LENGTH(DESTNAME)] <> ':' THEN
    IF DESTNAME[LENGTH(DESTNAME)] <> '.' THEN
      DESTNAME := CONCAT(DESTNAME,'.TEXT')
    ELSE
      DELETE(DESTNAME,LENGTH(DESTNAME),1);
RESET(SOURCEFILE,SOURCENAME);
REWRITE(F,DESTNAME);
READSEGDICT;
CLOSE(F,LOCK);
END.
```

Chris Wilson

Ein Pascal-Textformatierer

Die meisten kommerziellen Textverarbeitungs-Systeme bestehen in Wirklichkeit aus zwei Programmen: einem Bildschirm-orientierten Editor und einem Textformatierer. Da das Pascal-Betriebssystem bereits über einen sehr guten Editor verfügt, fehlt zur Textverarbeitung in Pascal nur noch das Formatierprogramm.

In dem ausgezeichneten Buch *Software Tools* von Kernigham und Plauger findet sich eine Reihe nützlicher Programme, darunter auch ein Textformatierer. Ich habe deren FORMAT-Programm von Ratfor in Pascal übersetzt und diverse (von den Autoren vorgeschlagene) Erweiterungen vorgenommen. Das Programmlisting finden Sie am Ende dieses Kapitels.

Die von FORMAT zu verarbeitenden Dateien bestehen aus einer Mischung von Befehlszeilen (die das Ausgabeformat beschreiben) und Textzeilen. Jede Zeile, die mit einem Punkt beginnt (.), wird als Befehlszeile angesehen. Unverständliche Befehlszeilen werden einfach ignoriert.

Auf den Punkt einer Befehlszeile folgt ein aus zwei Buchstaben bestehender Befehl. Zu manchen Befehlen gibt es optionale Parameter, die vom Befehlsnamen durch mindestens ein Leerzeichen getrennt sein müssen. Bevor die Befehlsnamen interpretiert werden, werden sie Zeichen für Zeichen in Großbuchstaben umgewandelt; man kann sie daher als Groß- oder Kleinbuchstaben eingeben.

So sorgt beispielsweise der Befehl:

```
.in 10
```

für eine Einrückung von 10 Spalten.

Zur Angabe numerischer Parameter für Befehle wie ".in" gibt es mehrere Möglichkeiten. Die erste ist, den Parameter ganz wegzulassen, mit dem Resultat, daß der Formatierer einen Vorgabewert verwendet (beim .in-Befehl den Wert 0):

```
.in
```

Die zweite Möglichleit ist die direkte Parametereingabe:

```
.in 10
```

Die dritte Möglichkeit besteht darin, die (positive oder negative) Differenz zwischen dem aktuellen und dem gewünschten Wert einzugeben:

```
.in +5
.in -5
```

FORMAT verfügt über zwei Modi zur Verarbeitung von Textzeilen, nämlich den "Wortmodus" und den "Zeilenmodus".

Im Wortmodus werden die Textzeilen in Wörter zerlegt, die dann unter Beachtung von Randbegrenzungen in neuformatierte Ausgabezeilen umgespeichert werden. Die Ausgabezeilen werden vor der Ausgabe durch Hinzufügen von Leerzeichen rechtsbündig formatiert. Man bezeichnet diesen Prozess als "Füllen" (filling).

Im Zeilenmodus werden die Textzeilen ohne interne Umgruppierungen ausgegeben, wobei aber immer noch Formatierungsparameter wie Randbegrenzung, mittiges Schreiben oder Unterstreichungen berücksichtigt werden.

Der Füllmodus wird vorgabemäßig eingeschaltet und läßt sich mit dem Befehl:

```
.nf
```

abschalten (no filling). Mit

```
.fi
```

kann er wieder eingeschaltet werden. Wenn eine Ausgabezeile beim Ausschalten des Füllmodus erst teilweise gefüllt ist, wird diese Zeile sofort ausgegeben, bevor weitere Formatierkommandos berücksichtigt werden. Man nennt diesen Vorgang "Break" (Abbruch). Etliche Formatierbefehle implizieren ein Break; mit dem Befehl:

```
.br
```

kann man ein Break auch explizit auslösen.

Der Einrückungsbefehl gibt an, um wieviele Spalten die nachfolgenden Ausgabezeilen eingerückt werden sollen (linker Rand):

```
.in 10
```

Der "Temporary Indent"-Befehl (etwa "vorübergehende Einrückung") löst ein Break aus und bestimmt, um wieviele Spalten die nächste Zeile einzurücken ist:

```
.ti +5
```

Durch den Befehl "Right Margin" (rechter Rand), wird angegeben, bei welcher Spalte der rechte Rand des Textes liegen soll:

```
.rm 72
```

Der Füllmodus berücksichtigt die jeweils durch ".rm", ".ti" und ".in" festgelegte Textbegrenzung.

Der "Center"-Befehl erzeugt ein Break und sorgt dann dafür, daß die nachfolgenden Zeilen mittig geschrieben werden, und zwar solange, bis entweder eine zuvor angegebene Zeilenanzahl zentriert worden ist oder ein Zentrierbefehl mit dem Wert 0 auftritt. Der Befehl:

```
.ce
«Textzeile»
```

zentriert auf diese Weise eine Zeile, während

```
.ce n
«Textzeile 1»
«Textzeile 2»

 :
«Textzeile n»
```

insgesamt n Zeilen zentriert, die im voraus gezählt werden. Mit

```
.ce 500
«Textzeile 1»
«Textzeile 2»

 :
«Textzeile n»
.ce 0
```

lassen sich n Zeilen zentrieren, die nicht im voraus gezählt wurden (mit n « = 500).

Nach Aufruf des "Underline"(Unterstreichungs)-Befehls werden alle Zeilen unterstrichen, bis entweder die zuvor angegebene Zeilenanzahl erreicht ist oder bis ein Unterstreichnungbefehl mit dem Wert 0 gefunden wird:

.ul 2
«Textzeile 1»
«Textzeile 2»

Dieser Befehl ähnelt dem Zentrierbefehl mit dem Unterschied, daß hier kein Break ausgelöst wird, so daß man im Füllmodus einzelne Wörter unterstreichen kann:

«Textzeile»
.ul
« zu unterstreichender Text»
«Textzeile»
«Textzeile»

Man kann Zentrier- und Unterstreichungsbefehle auch miteinander verbinden:

.ce
.ul
«Zu zentrierender und zu unterstreichender Text»

Zwischen zwei Ausgabezeilen wird automatisch eine Textzeile freigelassen. Der "Line Space"(Zeilenvorschub)-Befehl dient dazu, den Zeilenvorschub zwischen den Ausgabezeilen zu ändern. Gibt man den Wert "2" ein, so wird der Zeilenvorschub verdoppelt:

.ls 2

Der "Space"(Leerzeilen)-Befehl erzeugt ein Break und gibt dann Leerzeilen aus, bis die angegebene Anzahl leerer Zeilen erzeugt wurde oder ein Seitenende erreicht ist:

.sp 2

Leerzeilen, die durch einen Zeilenvorschub von mehr als eins erzeugt werden, entstehen als ein Nebenprodukt beim Schreiben der aktuellen Ausgabezeile. Da bei einem Break die aktuelle Ausgabezeile ausgegeben wird, wenn sie nicht leer ist, kann ein Zeilenvorschub als Nebeneffekt eines Breaks auftreten. Es ist wichtig, dieses Phänomen zu verstehen, da hierdurch mehr

Leerzeilen erzeugt werden können, als man bei Anwendung des Space-Befehls erwarten würde.

Leere Textzeilen und Zeilen, die mit Leerzeichen beginnen, werden besonders behandelt, wodurch sich die Anzahl der explizierten Formatierbefehle verringert.

Beim Auftreten einer Leerzeile wird ein Break ausgelöst (mit dem entsprechenden Zeilenabstand für die aktuelle Ausgabezeile) und danach eine Leerzeile ausgegeben (ebenfalls mit dem entsprechenden Zeilenabstand).

Zeilen, die nicht leer sind, aber mit Leerzeichen beginnen, erzeugen ein Break und eine temporäre Einrückung, die der Anzahl der Leerzeichen entspricht, mit denen die Zeile beginnt.

Der "Need"-Befehl (etwa: "Reservierungs"-Befehl) bewirkt ein Break. Danach wird festgestellt, ob die angegebene Zahl von Ausgabezeilen noch auf die aktuelle Seite paßt; wenn nicht, wird ein Seitenvorschub erzeugt.

```
.ne 10
```

Beim "Need"-Befehl wird der Zeilenvorschub berücksichtigt. Damit wird ein Wert von 10 automatisch in 20 umgerechnet, wenn man doppelten Zeilenabstand gewählt hat.

Die Befehle "Header" (Seitenkopf) und "Footer" (Seitenabschluß, Fußnote) dienen dazu, die oberste und unterste Zeile für die nachfolgenden Seiten zu bestimmen.

```
.he/links/Mitte/rechts/
.fo/links/Mitte/rechts/
```

Das "/"-Zeichen ist ein frei wählbares Begrenzungszeichen, das dazu dient, die drei Teile eines laufenden Titels voneinander zu trennen. Der linke Teil wird linksbündig, der rechte Teil rechtsbündig und der mittlere Teil mittig geschrieben. Die dabei benutzten Randbreiten sind diejenigen, die zur Verarbeitungszeit des Befehls und nicht etwa zur Zeit des Titelausdrucks gültig sind.

Erscheint irgendwo in einem der drei Teile ein "#"-Zeichen, so wird es durch die aktuelle Seitennummer ersetzt. Jeder der drei Teile kann leer sein.

```
.he "25.Jul.84""Seite#"
```

Durch Angabe eines leeren Parameters wird der entsprechende Titel gelöscht:

```
.he
```

Der ''Begin Page''(Neue Seite)-Befehl bewirkt ein Break. Danach wird die aktuelle Seite (falls vorhanden) abgeschlossen (Ausdruck der Fußzeile usw.); danach wird mit der neuen Seite begonnen:

.bp

Durch einen Befehl wie:

.bp 45

wird die Seitennummer der nächsten Seite auf den entsprechenden Wert festgelegt.

Der ''Page Length''(Seitenlängen)-Befehl legt fest, wie viele Zeilen eine Seite haben soll:

.pl 66

In Tabelle 1 finden Sie das Seitenformat inclusive der Kopf- und Fußzeile.

Tab. 1: Seitenformat

Zeile	**Bedeutung**
1	
2	[Kopfzeile]
3	
4	
5	[1. Ausgabezeile]
6	[2. Ausgabezeile]
	:
:	:
pl-5	[vorletzte Ausgabezeile]
pl-4	[letzte Ausgabezeile]
pl-3	
pl-2	
pl-1	[Fußzeile]
pl	

Der ''Source''(Quelltext)-Befehl ermöglicht es, eine weitere Datei einzuschließen:

.so «Dateiname»

Dieser Befehl ähnelt der "Include"-Option des Pascal-Compilers.

Die letzten Befehle sind "Define" (Definieren) und "End Define" (Definitionsende). Mit ihnen können Sie eigene parametrische Befehle definieren, die sowohl aus Formatbefehlen als auch aus Textzeilen bestehen:

```
.de p1
.ne 4
.sp 1
.in 10
.rm 72
.ti +5
.en
```

Definitionsnamen müssen wie normale Befehlsnamen zwei Zeichen lang sein. Wenn Sie wollen, können Sie damit sogar bestehende Befehlsnamen umdefinieren. Die Definitionen werden anhand ihres Namens aufgerufen:

```
.p1
```

Innerhalb der Definition stehende Textparameter werden durch die Notation "$1", "$2" usw. bezeichnet. Die Zahl hinter dem Dollarzeichen gibt an, auf welchen Parameter Bezug genommen wird, und darf nur im Bereich 1 bis 9 liegen. Die Parameterreferenzen brauchen nicht durch Leerzeichen getrennt zu werden.

```
.de f1
.ti 60
$1
.sp 2
Lieber $2:
.sp 1
Vielen Dank für Ihren letzten Beitrag zu Call-A.P.P.L.E.
mit dem Titel:
.ce
.ul
.en
```

Wird eine parametrische Definition aufgerufen, so werden die aktuellen Parameter nach dem Definitionsnamen angegeben. Sollen in einem Parameter Leerzeichen auftreten, so kann man ihn auch in Hochkommata oder Anführungsstriche einschließen:

```
.f1 "26.Juli 1983" Chris
Ein Pascal-Textformatierprogramm
```

Nicht spezifizierte Parameter werden durch leere Strings ersetzt, überflüssige Parameter ignoriert.

Tabelle 2 enthält eine Übersicht über die verschiedenen Befehle und deren Vorgabewerte:

Tab. 2: Befehlsübersicht

Befehl	**Vorgabewert**	**Break**
.bp n	n = +1	ja
.br		ja
.ce n	n = 1	ja
.de		ja
.en		nein
.fi		ja
.fo	////	nein
.he	////	nein
.in n	n = 0	nein
.ls n	n = 1	nein
.ne n	n = 1	ja
.nf		ja
.pl n	n = 66	nein
.rm n	n = 80	nein
.so		nein
.sp n	n = 1	ja
.ti n	n = 0	ja
.ul n	n = 1	nein

Der Quelltext des Formatierprogramms ist zu lang, um als einzelne Datei editiert werden zu können. Er wurde daher in zwei Textdateien mit den Namen FORMAT.TEXT und FORMAT.1.TEXT zerlegt, die im folgenden in dieser Reihenfolge abgedruckt sind.

```
(*$S+*)

(*$C Copyright (c) 1981 Chris Wilson *)

PROGRAM FORMAT;

CONST
  HUGE = 1000;
  MAXDEFPOOL = 5000;
  PAGEWIDTH = 80;
  PAGELENGTH = 66;


TYPE
  STRING255 = STRING[255];
  ARGTYPE = (DEFAULTED,RELPLUS,RELMINUS,ABSOLUTE);
  CMDTYPE = (UNKNOWN,DEFINED,BP,BR,CE,DE,EN,FI,FO,
    HE,IND,LS,NE,NF,PL,RM,SO,SP,TI,UL);
  TITLEINFO = RECORD
    EMPTY: BOOLEAN;
    LEFTM,
    RIGHTM: INTEGER;
    LEFTS,
    CENTERS,
    RIGHTS: STRING255
    END;
  ENTRYP = ^ ENTRY;
  ENTRY = RECORD
    NAME: PACKED ARRAY [1..8] OF CHAR;
    LLINK,
    RLINK: ENTRYP;
    TYP: CMDTYPE;
    START,
    LINES: INTEGER
    END;

VAR
  BACKSPACE: CHAR;
  DOTCOUNT,
  LINECOUNT: INTEGER;
  EMITPAGE,
  INCLUDING,
  NONCONSOLE: BOOLEAN;
  INCLUDEFILE,
  SOURCEFILE,
  DESTFILE: TEXT;
  SOURCENAME,
  DESTNAME: STRING;
  FILL: BOOLEAN;
  LSVAL,  (* ZEILENABSTAND *)
  INVAL,  (* EINRUECKUNG *)
  RMVAL,  (* RECHTER RAND *)
  TIVAL,  (* TEMPORAERE EINRUECKUNG *)
  CEVAL,  (* ANZAHL DER ZU ZENTRIERENDEN ZEILEN *)
  ULVAL,  (* ANZAHL DER ZU UNTERSTREICHENDEN ZEILEN *)
  CURPAGE,  (* SEITENNUMMER *)
  NEWPAGE,  (* NAECHSTE SEITENNUMMER *)
  LINENO, (* NAECHSTE AUSZUGEBENDE ZEILE *)
  PLVAL,  (* SEITENLAENGE IN ZEILEN *)
  M1VAL,  (* OBERER RAND, INCL. KOPFZEILE *)
```

```
  M2VAL,  (* RAND NACH KOPFZEILE *)
  M3VAL,  (* RAND NACH DER LETZTEN TEXTZEILE *)
  M4VAL,  (* UNTERER RAND INCL. FUSSZEILE *)
  BOTTOM, (* LETZTE ZEILE AUF DER SEITE *)
  OUTW,   (* BREITE DES AKTUELL IN OUTBUF STEHENDEN TEXTES *)
  OUTWDS, (* ANZAHL DER IN OUTBUF BEFINDLICHEN WOERTER *)
  DIR: INTEGER;
  HEADER,
  FOOTER: TITLEINFO;
  HASNUMERICARG: SET OF CMDTYPE;
  INBUF,
  OUTBUF,
  BLANKS255: STRING255;
  UNDERLINE: STRING[2];
  ROOT: ENTRYP;
  POOLINX: INTEGER;
  DEFPOOL: PACKED ARRAY [0..MAXDEFPOOL] OF CHAR;

PROCEDURE COMMAND(VAR BUF: STRING255);
  FORWARD;

PROCEDURE UPSTRING(VAR S: STRING);
        (*========*)

VAR
  I: INTEGER;

BEGIN
FOR I := 1 TO LENGTH(S) DO
  IF S[I] IN ['a'..'z'] THEN
    S[I] := CHR(ORD('A')+(ORD(S[I])-ORD('a')));
END;

FUNCTION MIN(A, B: INTEGER): INTEGER;
       (*===*)

BEGIN
IF A < B THEN
  MIN := A
ELSE
  MIN := B;
END;

FUNCTION MAX(A, B: INTEGER): INTEGER;
       (*===*)

BEGIN
IF A < B THEN
  MAX := B
ELSE
  MAX := A;
END;

PROCEDURE ERROR(S: STRING255);
        (*=====*)

BEGIN
WRITELN;
WRITELN('>>> ',S,' <<<');
CLOSE(DESTFILE,LOCK);
```

```
WRITELN(LINECOUNT,' Zeilen');
EXIT(FORMAT);
END;

FUNCTION GETLINE(VAR INBUF: STRING255): BOOLEAN;
       (*=======*)

BEGIN
IF INCLUDING THEN
  IF EOF(INCLUDEFILE) THEN
    BEGIN
    CLOSE(INCLUDEFILE);
    INCLUDING := FALSE;
    GETLINE := GETLINE(INBUF);
    EXIT(GETLINE);
    END
  ELSE
    READLN(INCLUDEFILE,INBUF)
ELSE IF EOF(SOURCEFILE) THEN
  BEGIN
  INBUF := '';
  GETLINE := FALSE;
  EXIT(GETLINE);
  END
ELSE
  READLN(SOURCEFILE,INBUF);
LINECOUNT := SUCC(LINECOUNT);
IF NONCONSOLE THEN
  BEGIN
  DOTCOUNT := SUCC(DOTCOUNT);
  WRITE('.');
  IF DOTCOUNT >= 50 THEN
    BEGIN
    WRITELN;
    WRITE('<',LINECOUNT:4,'>');
    DOTCOUNT := 0;
    END;
  END;
GETLINE := TRUE;
END;

FUNCTION ENTERCMD(CMD: STRING255; CT: CMDTYPE): ENTRYP;
       (*========*)

(* BEFEHL IN TABELLE EINFUEGEN *)

VAR
  P,
  Q: ENTRYP;
  I: INTEGER;

BEGIN
UPSTRING(CMD);
IF LENGTH(CMD) < 2 THEN
  ERROR(CONCAT('Befehlsname zu kurz: ',CMD));
NEW(P);
WITH P^ DO
  BEGIN
  NAME := '        ';
  MOVELEFT(CMD[1],NAME[1],2);
```

```
  LLINK := NIL;
  RLINK := NIL;
  END;
 IF ROOT = NIL THEN
   ROOT := P
 ELSE
  BEGIN
  I := TREESEARCH(ROOT,Q,P^.NAME);
  IF I = 1 THEN
    Q^.RLINK := P
  ELSE IF I = -1 THEN
    Q^.LLINK := P
  ELSE
    P := Q; (* NEUDEFINIEREN *)
  END;
P^.TYP := CT;
ENTERCMD := P;
END;

FUNCTION LOOKUP(VAR TOKEN: STRING255): ENTRYP;
       (*======*)

(* TABELLE NACH BEFEHL DURCHSUCHEN *)

VAR
  P: ENTRYP;
  NAM: PACKED ARRAY [1..8] OF CHAR;

BEGIN
LOOKUP := NIL;
IF LENGTH(TOKEN) >= 2 THEN
  BEGIN
  NAM := '        ';
  MOVELEFT(TOKEN[1],NAM[1],2);
  IF TREESEARCH(ROOT,P,NAM) = 0 THEN
    LOOKUP := P;
  END;
END;

PROCEDURE SETVAL(VAR PARAM: INTEGER; VAL: INTEGER; TYP: ARGTYPE;
        (*======*)   DEFVAL, MINVAL, MAXVAL: INTEGER);

(* PARAMETER EINSTELLEN UND WERTEBEREICH PRUEFEN *)

BEGIN
CASE TYP OF
DEFAULTED:
  PARAM := DEFVAL;
RELPLUS:
  PARAM := PARAM+VAL;
RELMINUS:
  PARAM := PARAM-VAL;
ABSOLUTE:
  PARAM := VAL
END; (* CASE *)
PARAM := MIN(PARAM,MAXVAL);
PARAM := MAX(PARAM,MINVAL);
END;
```

```
(*$I FORMAT.1 *)

PROCEDURE HANDLEARGS(VAR S, ARGS: STRING255);
        (*==========*)

(* AUFRUFARGUMENTE VON MAKROS BEARBEITEN *)

VAR
  I: INTEGER;
  ARG: STRING255;

PROCEDURE GETARG(N: INTEGER; VAR ARG: STRING255);
  VAR I: INTEGER;
    A: STRING255;
  BEGIN
  A := ARGS;
  WHILE (N > 0) AND (LENGTH(A) > 0) DO
    BEGIN
    IF (A[1] = '''') AND (LENGTH(A) > 1) THEN
      BEGIN
      DELETE(A,1,1);
      I := SCAN(LENGTH(A),='''',A[1]);
      ARG := COPY(A,1,I);
      DELETE(A,1,I);
      IF POS('''',A) = 1 THEN
        DELETE(A,1,1);
      END
    ELSE IF (A[1] = '"') AND (LENGTH(A) > 1) THEN
      BEGIN
      DELETE(A,1,1);
      I := SCAN(LENGTH(A),='"',A[1]);
      ARG := COPY(A,1,I);
      DELETE(A,1,I);
      IF POS('"',A) = 1 THEN
        DELETE(A,1,1);
      END
    ELSE
      BEGIN
      I := SCAN(LENGTH(A),=' ',A[1]);
      ARG := COPY(A,1,I);
      DELETE(A,1,I);
      END;
    IF LENGTH(A) > 0 THEN
      BEGIN
      I := SCAN(LENGTH(A),<>' ',A[1]);
      DELETE(A,1,I);
      END;
    N := PRED(N);
    END;
  IF N <> 0 THEN
    ARG := '';
  END;

BEGIN (* HANDLEARGS *)
I := 1;
WHILE I < LENGTH(S) DO
  BEGIN
  I := I+SCAN(LENGTH(S)+1-I,='$',S[I]);
  IF I < LENGTH(S) THEN
```

```
    IF S[I+1] IN ['1'..'9'] THEN
      BEGIN
      GETARG(ORD(S[I+1])-ORD('0'),ARG);
      DELETE(S,I,2);
      IF LENGTH(S)+LENGTH(ARG) <= 255 THEN
        INSERT(ARG,S,I)
      ELSE
        ERROR('Ueberlauf bei Makrosubstitution');
      I := I+LENGTH(ARG);
      END
    ELSE
      I := SUCC(I)
  ELSE
    I := SUCC(I);
  END;
END;

FUNCTION COMTYPE(VAR BUF: STRING255; EXPAND: BOOLEAN): CMDTYPE;
       (*=======*)

(* BEFEHL DECODIEREN UND AUS PUFFER LOESCHEN *)

VAR
  I,
  J: INTEGER;
  P: ENTRYP;
  S,
  CMDBUF: STRING255;

BEGIN
I := SCAN(LENGTH(BUF),=' ',BUF[1]);
CMDBUF := COPY(BUF,1,I);
DELETE(BUF,1,I);
IF LENGTH(BUF) > 0 THEN
  BEGIN
  I := SCAN(LENGTH(BUF),<>' ',BUF[1]);
  DELETE(BUF,1,I);
  END;
DELETE(CMDBUF,1,1); (* '.' *)
UPSTRING(CMDBUF);
P := LOOKUP(CMDBUF);
IF P = NIL THEN
  COMTYPE := UNKNOWN
ELSE
  WITH P^ DO
BEGIN
COMTYPE := TYP;
IF (TYP = DEFINED) AND EXPAND THEN
  BEGIN
  J := START;
  FOR I := 1 TO LINES DO
    BEGIN
    (* TRICKREICHER BEFEHL, BENUTZT LAENGENBYTE VON S *)
    MOVELEFT(DEFPOOL[J],S,1+ORD(DEFPOOL[J]));
    J := J+1+LENGTH(S);
    IF POS('$',S) <> 0 THEN
      HANDLEARGS(S,BUF);
    IF POS('.',S) = 1 THEN
      COMMAND(S)
```

```
        ELSE
          TXT(S);
        END;
      END;
    END;
END;

FUNCTION GETVAL(VAR BUF: STRING255; VAR TYP: ARGTYPE): INTEGER;
       (*======*)

(* ERMITTLE OPTIONALES NUMERISCHES ARGUMENT *)

VAR
  I: INTEGER;
  CONTINUE: BOOLEAN;

BEGIN
IF LENGTH(BUF) = 0 THEN
  TYP := DEFAULTED
ELSE IF POS('+',BUF) = 1 THEN
  TYP := RELPLUS
ELSE IF POS('-',BUF) = 1 THEN
  TYP := RELMINUS
ELSE
  TYP := ABSOLUTE;
IF TYP IN [RELPLUS,RELMINUS] THEN
  DELETE(BUF,1,1);
I := 0;
CONTINUE := TRUE;
WHILE CONTINUE AND (LENGTH(BUF) > 0) DO
  IF BUF[1] IN ['0'..'9'] THEN
    BEGIN
    I := (I*10)+(ORD(BUF[1])-ORD('0'));
    DELETE(BUF,1,1);
    END
  ELSE
    CONTINUE := FALSE;
GETVAL := I;
END;

PROCEDURE GETTL(VAR BUF: STRING255; VAR TITLE: TITLEINFO);
        (*=====*)

(* TITEL AUS PUFFER HOLEN *)

PROCEDURE GETPART(VAR BUF, S: STRING255);
  VAR I: INTEGER;
    DELIM: STRING[1];
  BEGIN
  IF LENGTH(BUF) > 0 THEN
    BEGIN
    DELIM := COPY(BUF,1,1);
    DELETE(BUF,1,1);
    I := POS(DELIM,BUF);
    IF I = 0 THEN
      BEGIN
      S := BUF;
      BUF := '';
      END
```

```
    ELSE
      BEGIN
      I := PRED(I);
      S := COPY(BUF,1,I);
      DELETE(BUF,1,I);
      END;
    END
  ELSE
        BEGIN
        I := PRED(I);
        S := COPY(BUF,1,I);
        DELETE(BUF,1,I);
        END;
      END
    ELSE
      S := '';
    END;

BEGIN (* GETTL *)
WITH TITLE DO
  BEGIN
  LEFTM := INVAL;
  RIGHTM := RMVAL;
  GETPART(BUF,LEFTS);
  GETPART(BUF,CENTERS);
  GETPART(BUF,RIGHTS);
  EMPTY := (LEFTS = '') AND (CENTERS = '') AND (RIGHTS = '');
  END;
END;

PROCEDURE DEFINE(VAR BUF: STRING255);
        (*======*)

(* MAKRO DEFINIEREN *)

VAR
  P: ENTRYP;
  CT: CMDTYPE;
  INBUF: STRING255;

BEGIN
P := ENTERCMD(BUF,DEFINED);
WITH P^ DO
  BEGIN
  START := POOLINX;
  LINES := 0;
  WHILE GETLINE(INBUF) DO
    IF POOLINX+1+LENGTH(INBUF) <= MAXDEFPOOL THEN
      BEGIN
      (* TRICKREICHER BEFEHL, BENUTZT LAENGENBYTE VON INBUF *)
      MOVELEFT(INBUF,DEFPOOL[POOLINX],1+LENGTH(INBUF));
      POOLINX := POOLINX+1+LENGTH(INBUF);
      LINES := SUCC(LINES);
      IF POS('.',INBUF) = 1 THEN
        BEGIN
        CT := COMTYPE(INBUF,FALSE);
        IF CT = DE THEN
          ERROR(CONCAT('Geschachtelte Definitionen verboten: ',BUF))
        ELSE IF CT = EN THEN
```

```
          BEGIN
          LINES := PRED(LINES);
          EXIT(DEFINE);
          END;
        END
      END
    ELSE
      ERROR(CONCAT('Definition zu lang: ',BUF));
  END;
END;

PROCEDURE COMMAND; (* VAR BUF: STRING255 *)
        (*=======*)

(* FORMATIERBEFEHL AUSFUEHREN *)

VAR
  VAL,
  NEVAL,
  SPVAL: INTEGER;
  CT: CMDTYPE;
  TYP: ARGTYPE;

BEGIN
CT := COMTYPE(BUF,TRUE);
IF CT IN HASNUMERICARG THEN
  VAL := GETVAL(BUF,TYP);
CASE CT OF
UNKNOWN, DEFINED:
  BEGIN
  END;
BP:
  BEGIN
  IF LINENO > 0 THEN
    SPACE(HUGE);
  SETVAL(CURPAGE,VAL,TYP,SUCC(CURPAGE),-HUGE,HUGE);
  NEWPAGE := CURPAGE;
  END;
BR:
  BRK;
CE:
  BEGIN
  BRK;
  SETVAL(CEVAL,VAL,TYP,1,0,HUGE);
  END;
DE:
  BEGIN
  BRK;
  DEFINE(BUF);
  END;
EN:
  ERROR('.en ausserhalb von Definition');
FI:
  BEGIN
  BRK;
  FILL := TRUE;
  END;
FO:
  GETTL(BUF,FOOTER);
```

```
HE:
  GETTL(BUF,HEADER);
IND:
  BEGIN
  SETVAL(INVAL,VAL,TYP,0,0,PRED(RMVAL));
  TIVAL := INVAL;
  END;
LS:
  SETVAL(LSVAL,VAL,TYP,1,1,HUGE);
NE:
  BEGIN
  BRK;
  NEVAL := 1; (* FALLS VAL RELATIV IST *)
  SETVAL(NEVAL,VAL,TYP,1,0,HUGE);
  NEVAL := NEVAL*LSVAL;
  IF LINENO+PRED(NEVAL) > BOTTOM THEN
    SPACE(HUGE);
  END;
NF:
  BEGIN
  BRK;
  FILL := FALSE;
  END;
PL:
  BEGIN
  SETVAL(PLVAL,VAL,TYP,PAGELENGTH,M1VAL+M2VAL+M3VAL+M4VAL+1,HUGE);
  BOTTOM := PLVAL-M3VAL-M4VAL;
  END;
RM:
  SETVAL(RMVAL,VAL,TYP,PAGEWIDTH,SUCC(TIVAL),255);
SO:
  BEGIN
  UPSTRING(BUF);
  IF BUF = '' THEN
    EXIT(COMMAND)
  ELSE IF POS('.TEXT',BUF) = 0 THEN
    IF BUF[LENGTH(BUF)] <> ':' THEN
      IF BUF[LENGTH(BUF)] <> '.' THEN
        BUF := CONCAT(BUF,'.TEXT')
      ELSE
        DELETE(BUF,LENGTH(BUF),1);
  IF INCLUDING THEN
    ERROR('Fileschachtelung verboten')
  ELSE
    BEGIN
    INCLUDING := TRUE;
    RESET(INCLUDEFILE,BUF);
    END;
  END;

SP:
  BEGIN
  SPVAL := 1; (* FALLS VAL RELATIV IST *)
  SETVAL(SPVAL,VAL,TYP,1,0,HUGE);
  SPACE(SPVAL);
  END;
```

```
TI:
  BEGIN
  BRK;
  SETVAL(TIVAL,VAL,TYP,0,0,RMVAL);
  END;
UL:
  SETVAL(ULVAL,VAL,TYP,1,0,HUGE)
END; (* CASE *)
END;

PROCEDURE INIT;
        (*====*)

VAR
  P: ENTRYP;

BEGIN
BACKSPACE := CHR(8);
DOTCOUNT := 0;
LINECOUNT := 0;
INCLUDING := FALSE;
FILL := TRUE;
LSVAL := 1;
INVAL := 0;
RMVAL := PAGEWIDTH;
TIVAL := 0;
CEVAL := 0;
ULVAL := 0;
CURPAGE := 0;
NEWPAGE := 1;
LINENO := 0;
PLVAL := PAGELENGTH;
M1VAL := 2;
M2VAL := 2;
M3VAL := 2;
M4VAL := 2;
BOTTOM := PLVAL-M3VAL-M4VAL;
OUTW := 0;
OUTWDS := 0;
DIR := 0;
HEADER.EMPTY := TRUE;
FOOTER.EMPTY := TRUE;
HASNUMERICARG := [BP,CE,IND,LS,NE,PL,RM,SP,TI,UL];
OUTBUF := '';
(*$R-*)
BLANKS255[0] := CHR(255);
(*$R+*)
FILLCHAR(BLANKS255[1],255,' ');
UNDERLINE := '_ ';
UNDERLINE[2] := BACKSPACE;
ROOT := NIL;
POOLINX := 0;
P := ENTERCMD('BP',BP);
P := ENTERCMD('BR',BR);
P := ENTERCMD('CE',CE);
P := ENTERCMD('DE',DE);
P := ENTERCMD('EN',EN);
P := ENTERCMD('FI',FI);
P := ENTERCMD('FO',FO);
P := ENTERCMD('HE',HE);
```

```
P := ENTERCMD('NF',NF);
P := ENTERCMD('PL',PL);
P := ENTERCMD('RM',RM);
P := ENTERCMD('SO',SO);
P := ENTERCMD('SP',SP);
P := ENTERCMD('TI',TI);
P := ENTERCMD('UL',UL);
END;


(* ========= HAUPTPROGRAMM ========== *)

BEGIN
INIT;
PAGE(OUTPUT);
WRITELN('Format (27.7.81)');
WRITELN('Copyright (c) 1981 Chris Wilson');
WRITELN;
WRITE('Textdatei: ');
READLN(SOURCENAME);
UPSTRING(SOURCENAME);
IF SOURCENAME = '' THEN
  EXIT(FORMAT)
ELSE IF POS('.TEXT',SOURCENAME) = 0 THEN
  IF SOURCENAME[LENGTH(SOURCENAME)] <> ':' THEN
    IF SOURCENAME[LENGTH(SOURCENAME)] <> '.' THEN
      SOURCENAME := CONCAT(SOURCENAME,'.TEXT')
    ELSE
      DELETE(SOURCENAME,LENGTH(SOURCENAME),1);
WRITE('Ausgabedatei: ');
READLN(DESTNAME);
UPSTRING(DESTNAME);
IF DESTNAME = '' THEN
  DESTNAME := 'PRINTER:'
ELSE IF POS('.TEXT',DESTNAME) = 0 THEN
  IF DESTNAME[LENGTH(DESTNAME)] <> ':' THEN
    IF DESTNAME[LENGTH(DESTNAME)] <> '.' THEN
      DESTNAME := CONCAT(DESTNAME,'.TEXT')
    ELSE
      DELETE(DESTNAME,LENGTH(DESTNAME),1);
NONCONSOLE := (POS('CONSOLE:',DESTNAME) = 0)
  AND (POS('SYSTERM:',DESTNAME) = 0)
  AND (POS('#1:',DESTNAME) = 0)
  AND (POS('#2:',DESTNAME) = 0);
EMITPAGE := (POS('PRINTER:',DESTNAME) = 1)
  OR (POS('#6:',DESTNAME) = 1)
  OR NOT NONCONSOLE;
RESET(SOURCEFILE,SOURCENAME);
REWRITE(DESTFILE,DESTNAME);
IF NONCONSOLE THEN
  WRITE('<',LINECOUNT:4,'>');
WHILE GETLINE(INBUF) DO
  IF POS('.',INBUF) = 1 THEN
    COMMAND(INBUF)
  ELSE
    TXT(INBUF);
IF LINENO > 0 THEN
  SPACE(HUGE);
```

```
CLOSE(DESTFILE,LOCK);
IF NONCONSOLE THEN
  WRITELN;
WRITELN(LINECOUNT,' Zeilen');
END.
```

```
(* Copyright (c) 1981 by Chris Wilson *)

PROCEDURE SKIP(N: INTEGER);
        (*====*)

(* AUSGABE VON N LEERZEILEN *)

VAR
  I: INTEGER;

BEGIN
FOR I := 1 TO N DO
  WRITELN(DESTFILE);
END;

PROCEDURE PUTTL(VAR TITLE: TITLEINFO; PAGENO: INTEGER);
        (*=====*)

(* GIB TITELZEILE MIT OPTIONALER SEITENNUMMER AUS *)

VAR
  I,
  J: INTEGER;
  S: STRING255;
  PAGES: STRING;


PROCEDURE STR(VAL: INTEGER; VAR S: STRING);
  VAR D, I: INTEGER;
    MINUS: BOOLEAN;
  BEGIN
  MINUS := VAL < 0;
  VAL := ABS(VAL);
  S := '      ';
  I := 6;
  REPEAT
    D := VAL MOD 10;
    VAL := VAL DIV 10;
    S[I] := CHR(ORD('0')+D);
    I := PRED(I)
  UNTIL VAL = 0;
  IF MINUS THEN
    BEGIN
    S[I] := '-';
    I := PRED(I);
    END;
  DELETE(S,1,I);
  END;
```

```
PROCEDURE REPLACE(VAR S: STRING255);
  VAR I: INTEGER;
  BEGIN
  REPEAT
    I := POS('#',S);
    IF I <> 0 THEN
      BEGIN
      DELETE(S,I,1);
      INSERT(PAGES,S,I);
      END
  UNTIL I = 0;
  END;

BEGIN (* PUTTL *)
WITH TITLE DO
  IF NOT EMPTY THEN
    BEGIN
    STR(PAGENO,PAGES);
    IF LEFTM > 0 THEN
      WRITE(DESTFILE,' ':LEFTM);
    I := LEFTM;
    IF LENGTH(LEFTS) > 0 THEN
      BEGIN
      S := LEFTS;
      REPLACE(S);
      WRITE(DESTFILE,S);
      I := I+LENGTH(S);
      END;
    IF LENGTH(CENTERS) > 0 THEN
      BEGIN
      S := CENTERS;
      REPLACE(S);
      J := MAX(((LEFTM+RIGHTM)-LENGTH(S)) DIV 2,0);
      IF I < J THEN
        WRITE(DESTFILE,' ':(J-I));
      WRITE(DESTFILE,S);
      I := J+LENGTH(S);
      END;
    IF LENGTH(RIGHTS) > 0 THEN
      BEGIN
      S := RIGHTS;
      REPLACE(S);
      J := RIGHTM-LENGTH(S);
      IF I < J THEN
        WRITE(DESTFILE,' ':(J-I));
      WRITE(DESTFILE,S);
      END;
    END;
WRITELN(DESTFILE);
END;

PROCEDURE PHEAD;
        (*=====*)

(* KOPFZEILE AUSGEBEN *)

BEGIN
CURPAGE := NEWPAGE;
NEWPAGE := SUCC(NEWPAGE);
```

```
IF EMITPAGE THEN
  PAGE(DESTFILE);
IF M1VAL > 0 THEN
  BEGIN
  SKIP(PRED(M1VAL));
  PUTTL(HEADER,CURPAGE);
  END;
SKIP(M2VAL);
LINENO := M1VAL+M2VAL+1;
END;

PROCEDURE PFOOT;
        (*=====*)

(* FUSSZEILE AUSGEBEN *)

BEGIN
SKIP(M3VAL);
IF M4VAL > 0 THEN
  BEGIN
  PUTTL(FOOTER,CURPAGE);
  SKIP(PRED(M4VAL));
  END;
END;

PROCEDURE PUT(VAR BUF: STRING255);
        (*===*)

(* ZEILE UNTER BEACHTUNG VON ZEILENABSTAND UND EINRUECKUNG AUSGEBEN *)

BEGIN
IF (LINENO = 0) OR (LINENO > BOTTOM) THEN
  PHEAD;
IF TIVAL > 0 THEN
  WRITE(DESTFILE,' ':TIVAL);
TIVAL := INVAL;
WRITELN(DESTFILE,BUF);
SKIP(MIN(PRED(LSVAL),BOTTOM-LINENO));
LINENO := LINENO+LSVAL;
IF LINENO > BOTTOM THEN
  PFOOT;
END;

PROCEDURE BRK;
        (*===*)

(* AKTUELLE ZEILE AUSGEBEN *)

BEGIN
IF LENGTH(OUTBUF) > 0 THEN
  PUT(OUTBUF);
OUTW := 0;
OUTWDS := 0;
OUTBUF := '';
END;

PROCEDURE SPACE(N: INTEGER);
        (*=====*)
```

```
(* N LEERZEILEN AUSGEBEN ODER UNTEREN SEITENRAND EINSTELLEN *)

BEGIN
BRK;
IF LINENO <= BOTTOM THEN
  BEGIN
  IF LINENO = 0 THEN
    PHEAD;
  SKIP(MIN(N,(BOTTOM+1)-LINENO));
  LINENO := LINENO+N;
  IF LINENO > BOTTOM THEN
    PFOOT;
  END;
END;

PROCEDURE LEADBL(VAR BUF: STRING255);
        (*======*)

(* FUEHRENDE LEERZEICHEN LOESCHEN, TIVAL EINSTELLEN *)

VAR
  I: INTEGER;

BEGIN
BRK;
IF LENGTH(BUF) > 0 THEN
  BEGIN
  I := SCAN(LENGTH(BUF),<>' ',BUF[1]);
  DELETE(BUF,1,I);
  IF LENGTH(BUF) > 0 THEN
    TIVAL := I;
  END;
END;

FUNCTION WIDTH(VAR BUF: STRING255): INTEGER;
        (*=====*)

(* BERECHNE BREITE DES STRINGS *)

VAR
  I,
  BS,
  COUNT: INTEGER;

BEGIN
IF LENGTH(BUF) > 0 THEN
  BEGIN
  I := SCAN(LENGTH(BUF),=BACKSPACE,BUF[1]);
  IF I = LENGTH(BUF) THEN
    WIDTH := I
  ELSE
    BEGIN
    BS := 1;
    I := I+2;
    COUNT := (LENGTH(BUF)+1)-I;
    WHILE COUNT > 0 DO
      BEGIN
      I := I+SCAN(COUNT,=BACKSPACE,BUF[I]);
      COUNT := (LENGTH(BUF)+1)-I;
      IF COUNT <> 0 THEN
```

```
        BEGIN
        BS := SUCC(BS);
        I := SUCC(I);
        COUNT := PRED(COUNT);
        END;
      END;
    WIDTH := LENGTH(BUF)-BS;
    END;
  END
ELSE
  WIDTH := 0;
END;

PROCEDURE SPREAD(VAR BUF: STRING255; NEXTRA, OUTWDS: INTEGER);
        (*======*)

(* FUER RECHTEN RANDAUSGLEICH WOERTER AUSEINANDER RUECKEN *)

VAR
  I,
  NB,
  NE,
  NHOLES: INTEGER;

BEGIN
IF (NEXTRA > 0) AND (OUTWDS > 1) THEN
  BEGIN
  DIR := 1-DIR;
  NE := NEXTRA;
  NHOLES := PRED(OUTWDS);
  I := 1;
  WHILE NE > 0 DO
    BEGIN
    I := I+SCAN((LENGTH(BUF)+1)-I,=' ',BUF[I]);
    IF DIR = 0 THEN
      NB := (PRED(NE) DIV NHOLES)+1
    ELSE
      NB := NE DIV NHOLES;
    NE := NE-NB;
    NHOLES := PRED(NHOLES);
    INSERT(COPY(BLANKS255,1,NB),BUF,I);
    I := I+NB+1;
    END;
  END;
END;

PROCEDURE PUTWRD(VAR WRDBUF: STRING255);
        (*======*)

(* WORT IN OUTBUF SCHREIBEN; RANDAUSGLEICH BEACHTEN *)

VAR
  W,
  LAST,
  LLVAL: INTEGER;

BEGIN
W := WIDTH(WRDBUF);
LLVAL := RMVAL-TIVAL;
```

```
IF LENGTH(OUTBUF) > 0 THEN
  BEGIN
  LAST := LENGTH(OUTBUF)+1+LENGTH(WRDBUF);
  IF (OUTW+1+W > LLVAL) OR (LAST > 255) THEN
    BEGIN
    SPREAD(OUTBUF,LLVAL-OUTW,OUTWDS);
    BRK;
    END;
  END;
IF LENGTH(OUTBUF) = 0 THEN
  BEGIN
  OUTBUF := WRDBUF;
  OUTW := W;
  END
ELSE
  BEGIN
  INSERT(' ',OUTBUF,LENGTH(OUTBUF)+1);
  INSERT(WRDBUF,OUTBUF,LENGTH(OUTBUF)+1);
  OUTW := OUTW+1+W;
  END;
OUTWDS := SUCC(OUTWDS);
END;

PROCEDURE CENTER(VAR BUF: STRING255);
        (*======*)

(* ZEILE DURCH SETZEN VON TIVAL ZENTRIEREN *)

BEGIN
TIVAL := MAX(((RMVAL+TIVAL)-WIDTH(BUF)) DIV 2,0);
END;

PROCEDURE UNDERL(VAR BUF: STRING255);
        (*======*)

(* ZEILE UNTERSTREICHEN *)

VAR
  I: INTEGER;

BEGIN
I := 1;
WHILE (I < LENGTH(BUF)) AND (I < 254) DO
  BEGIN
  IF BUF[I] <> ' ' THEN
    BEGIN
    INSERT(UNDERLINE,BUF,I);
    I := I+2;
    END;
  I := SUCC(I);
  END;
END;

FUNCTION GETWRD(VAR INBUF, OUT: STRING255): INTEGER;
       (*======*)

(* NICHTLEERES WORT VON INBUF NACH OUT KOPIEREN UND AUS INBUF LOESCHEN *)
```

```
VAR
  I: INTEGER;

BEGIN
IF LENGTH(INBUF) > 0 THEN
  BEGIN
  I := SCAN(LENGTH(INBUF),<>' ',INBUF[1]);
  DELETE(INBUF,1,I);
  END;
IF LENGTH(INBUF) > 0 THEN
  BEGIN
  I := SCAN(LENGTH(INBUF),=' ',INBUF[1]);
  OUT := COPY(INBUF,1,I);
  DELETE(INBUF,1,I);
  GETWRD := I;
  END
ELSE
  BEGIN
  OUT := '';
  GETWRD := 0;
  END;
END;

PROCEDURE TXT(VAR INBUF: STRING255);
        (*===*)

(* TEXTZEILEN VERARBEITEN *)


VAR
  WRDBUF: STRING255;

BEGIN
IF LENGTH(INBUF) = 0 THEN
  LEADBL(INBUF)
ELSE IF INBUF[1] = ' ' THEN
  LEADBL(INBUF);
IF ULVAL > 0 THEN
  BEGIN
  UNDERL(INBUF);
  ULVAL := PRED(ULVAL);
  END;
IF CEVAL > 0 THEN
  BEGIN
  CENTER(INBUF);
  PUT(INBUF);
  CEVAL := PRED(CEVAL);
  END
ELSE IF LENGTH(INBUF) = 0 THEN
  PUT(INBUF)
ELSE IF NOT FILL THEN
  PUT(INBUF)
ELSE
  WHILE GETWRD(INBUF,WRDBUF) > 0 DO
    PUTWRD(WRDBUF);
END;
```

Dean Rosenhain

Ein Pascal-Zeicheneditor

Im APPLE-Pascal gibt es zwei Befehle, mit denen man Text auf der HI-RES-Grafikseite schreiben kann: WCHAR und WSTRING. Der von diesen Befehlen benutzte Zeichensatz ist in einem auf Diskette gespeicherten Datenfile namens SYSTEM.CHARSET gespeichert. Eine genaue Beschreibung des Datenformats dieser 1024 Bytes finden Sie auf den Seiten 99 und 100 des *APPLE Pascal Language Reference Manual*. Jedes Zeichen wird in acht aufeinanderfolgenden Bytes in einem Array von 128 Zeichen untergebracht. Die Pascal-Deklaration zur Erzeugung neuer Zeichensätze oder zur Modifikation bereits vorhandener sieht wie folgt aus:

```
TYPE
      BITS = 0..7;
      BYTE = SET OF BITS;
      CHARIMAGE = PACKED ARRAY [0..7] OF BYTE;
      CHARSET = PACKED ARRAY [0..127] OF CHARIMAGE;

VAR
      CHARACTERS : CHARSET;
      CHARFILE : FILE OF CHARSET;
```

Es gibt einen weiteren Weg, Textzeichen in der HI-RES-Grafik darzustellen, nämlich mit DRAWBLOCK, einer Turtlegraphics-Prozedur, die ein Bit-Array auf den Bildschirm kopiert und so die Darstellung von Bildern möglich macht. WCHAR und WSTRING benutzen die DRAWBLOCK-Prozedur (weitere Informationen über die Verwendung von Bit-Arrays finden Sie in Loy Spurlocks Artikel in *Applesauce*, Vol. 1, Heft 7 (Oktober 1979), der in *Call-A.P.P.L.E.* vom Januar 1980 nachgedruckt wurde).

Um zu erklären, wie DRAWBLOCK zur Zeichenausgabe benutzt werden kann, nehmen wir an, daß SYSTEM.CHARSET von einer Diskette gelesen und in ein Array namens CHARACTERS (mit der o.g. Typendeklaration)

gespeichert wurde. Will man z.B. das Zeichen "A" auf der Grafikseite ausgeben, so benutzt man dazu den Befehl:

DRAWBLOCK (CHARACTERS[65],1,0,0,7,8,140,90,10);

Das Array, das auf den Bildschirm kopiert wird, ist in diesem Fall CHARACTERS [65], da "A" den ASCII-Code 65 hat. Wie man aus der obigen Deklaration ersehen kann, besteht CHARACTERS [65] aus acht Bytes von CHARIMAGE. Die anderen Parameter geben folgendes an:

"1" = Anzahl der Bytes, die von jeder Punktreihe des Zeichens belegt werden.

"0" = Anzahl der Punkte oder Bits, die in der Zeile vor Beginn des zu kopierenden Arraybereiches übersprungen werden sollen.

"0" = Anzahl der Punkte, die in der Vertikalen übersprungen werden sollen.

"7" = Breite des Bildes in Punkten.

"8" = Höhe des Bildes in Punkten.

"140" = X-Koordinate der unteren linken Bildecke.

"90" = Y-Koordinate der unteren linken Bildecke.

"10" = Anzeigemodus.

Der Anzeigemodus ist ein nützlicher Parameter, mit dem man angeben kann, auf welche Weise das Bit-Array auf den Bildschirm kopiert werden soll (weitere Einzelheiten im Handbuch). Es gibt insgesamt 16 verschiedene Anzeigemodi, wie z.B. die inverse Darstellung (Modus Nr.5). Benutzt man Modus 6 (Exklusiv-ODER), kann man durch zweimaliges Schreiben des gleichen Textes an der gleichen Bildschirmstelle diesen Text wieder löschen, ohne daß der ursprüngliche Bildhintergrund zerstört wird. Der Standard-Schreibmodus ist Nr.10. Dabei werden die Bytes des Arrays direkt auf den Bildschirm kopiert.

Wenn es die DRAWBLOCK-Prozedur nicht gäbe, wäre es schwierig (und langwierig), Zeichensätze auf den Bildschirm zu kopieren, die nicht in SYSTEM.CHARSET stehen. Mit DRAWBLOCK dagegen kann man sogar mehrere Zeichensätze gleichzeitig in ein- und demselben Programm benut-

zen. Ohne allzu große Schwiereigkeiten kann man auch größere Zeichen definieren und die DRAWBLOCK-Parameter ihrer Ausgabe entsprechend anpassen. Wenn die Zeichen mit SYSTEM.CHARSET kompatibel sein sollen, müssen sie natürlich den obigen Deklarationen entsprechen.

Der hier vorgestellte Zeichensatzeditor wurde so entworfen, daß seine Benutzung recht einfach erlernt werden kann. Obwohl das Programmlisting halbwegs selbsterklärend ist, sind einige Eigenschaften des Programms und seiner Funktionsweise erwähnenswert.

Bei Programmstart zeigt die Prozedur SETUP den aktuellen Zeichensatz (der anfangs nur aus Zufallspunkten besteht). Danach gibt die Prozedur MENU1 einige der zur Verfügung stehenden Befehle aus. Durch Drücken der Leertaste wird MENU2 aufgerufen und der andere Teil des Befehlssatzes angezeigt. Die INPUT-Prozedur zeigt jeden Tastendruck auf dem HI-RES-Bildschirm an. Anstelle des Dateinamens "SYSTEM.CHARSET" kann man auch "*" angeben. Mit FILEIN werden Datenfiles gelesen und ggf. auftretende E/A-Fehler abgefangen. Die entsprechende Prozedur zum Speichern von Zeichensätzen heißt FILEOUT.

Das Zeichen, das editiert werden soll, wird mit einem quadratischen Cursor ausgewählt (HILITE-Prozedur), der sich mit den Tasten "W", "A", "S" und "Z" über den Zeichensatz bewegen läßt. Mit dem GRAB-Befehl kann man das so gewählte Zeichen im vergrößerten Maßstab auf der rechten Bildseite sichtbar machen. Die Tasten "I", "J", "K" und "M" verschieben das Fadenkreuz, das zum Setzen und Löschen einzelner Punkte dient. Mit "Y" und "N" läßt sich der durch das Kreuz bezeichnete Punkt setzen bzw. löschen. Mit dem Push-Befehl wird das geänderte Zeichen bei der Position des HILITE-Cursors in den Zeichensatz zurückkopiert.

Wenn Sie Ihren eigenen Zeichensatz erstellt haben, können Sie ihn in SYSTEM.CHARSET umbenennen und mit Hilfe der Turtlegraphics-Routinen WCHAR und WSTRING auf ihn zugreifen. Dazu müssen Sie den Filer aufrufen und die SYSTEM.CHARSET-Datei umbenennen, z.B.:

```
Change?APPLE1:SYSTEM.CHARSET
To?APPLE1:OLD.CHARSET
```

Danach benennen Sie Ihren eigenen Zeichensatz in "SYSTEM.CHARSET" um, damit er vom Betriebssystem gelesen werden kann. Das erreicht man zum Beispiel durch:

```
Transfer?MYDISK:MYSET
To?APPLE1:SYSTEM.CHARSET
```

oder, wenn sich der Zeichensatz schon auf der Diskette befindet:

Change?MYSET
To?SYSTEM.CHARSET

Von diesem Augenblick an können die Turtlegraphics-Prozeduren auf Ihren Zeichensatz zugreifen.

```
PROGRAM CHARED;
(************************************)
(*                                  *)
(*        HI-RES ZEICHEN            *)
(*            EDITOR                *)
(*         VERSION 2.2.1            *)
(*                                  *)
(*   VON... Dean  Rosenhain         *)
(*                                  *)
(************************************)

USES TURTLEGRAPHICS;

TYPE BITS = 0..7;
     BYTE = SET OF BITS;
     CHARIMAGE= PACKED ARRAY[0..7] OF BYTE;
     CHARSET= PACKED ARRAY[0..127] OF CHARIMAGE;

VAR  CURRENTSET:CHARSET;
     CURRENTCH :CHARIMAGE;
     CHARFILE  :FILE OF CHARSET;
     DISKIO,I,OLD,H,V,OLDH,OLDV:INTEGER;
     CHOICE:CHAR;
     ONE:BOOLEAN;

PROCEDURE PRINTAT(X,Y:INTEGER; S:STRING);
BEGIN
  MOVETO(X,Y);
  WSTRING(S);
END;

PROCEDURE INPUT(X,Y:INTEGER;VAR S:STRING);

(* Mit dieser Prozedur koennen Strings  *)
(* auf der Graphikseite eingegeben      *)
(* werden. Dabei wird die Moeglichkeit  *)
(* gegeben, Tippfehler mit der Links-   *)
(* pfeiltaste zu korrigieren.           *)
VAR TEMP :STRING;
    S1   :STRING[1];
    CH   :CHAR;
BEGIN
   TEMP:='';
   S1:=' ';
```

```
   MOVETO(X,Y);
   REPEAT
      READ(KEYBOARD,CH);
      IF CH IN ['!'..'Z'] THEN
      BEGIN
        WCHAR(CH);
        S1[1]:=CH;
        TEMP:=CONCAT(TEMP,S1);
      END
      ELSE
        IF (ORD(CH)=8) AND (LENGTH(TEMP)>0) THEN
        BEGIN
          DELETE(TEMP,LENGTH(TEMP),1);
          MOVETO(TURTLEX-7,Y);
          WCHAR(' ');
          MOVETO(TURTLEX-7,Y);
        END;
   UNTIL CH=CHR(32);
   S:=TEMP;
END;

PROCEDURE SHOWCHAR(NUM:INTEGER);
VAR X,Y:INTEGER;
BEGIN
  X:=10*(NUM MOD 16)+5;
  Y:=150-(NUM DIV 16)*10;
  DRAWBLOCK(CURRENTSET[NUM],1,0,0,7,8,X,Y,10);
END;

PROCEDURE DISPLAYSET;
VAR NUM:INTEGER;
BEGIN
    FOR NUM:= 0 TO 127 DO
     SHOWCHAR(NUM);
END;

PROCEDURE DISKERROR(ERR:INTEGER);
VAR S:STRING;
 TIME:INTEGER;
BEGIN
  CHARTYPE(5); (* INVERSE DARSTELLUNG *)
  IF ERR IN [6,7,10] THEN
   PRINTAT(10,2,'falscher Dateiname, nicht verfuegbar')
  ELSE
    BEGIN
      STR(ERR,S);
      S:=CONCAT('Diskettenfehler Nr. ',S);
      PRINTAT(10,2,S);
    END;

  WRITELN(CHR(7));
  FOR TIME:= 1 TO 3000 DO (* nichts *);
  CHARTYPE(10);  (* Normalmodus *)
  WRITELN(CHR(7));
  PRINTAT(10,2,'                                    ');
END;

PROCEDURE FILEIN;
VAR FILENAME:STRING;
```

```
BEGIN
    INPUT(101,27,FILENAME);
    PRINTAT(101,27,'                    ');
    IF LENGTH(FILENAME)=0 THEN EXIT(FILEIN);
    IF FILENAME='*' THEN FILENAME:='SYSTEM.CHARSET';
(*$I-*)
    RESET(CHARFILE,FILENAME);
    DISKIO:=IORESULT;
    IF DISKIO<>0 THEN DISKERROR(DISKIO)
    ELSE
    BEGIN
      CURRENTSET:= CHARFILE^;
      CLOSE(CHARFILE,LOCK);
      DISPLAYSET;
      PRINTAT(90,160,'                    ');
      PRINTAT(90,160,FILENAME);
    END;
(*$I+*)
END;

PROCEDURE HILITE(LETTER:INTEGER; COLOR:SCREENCOLOR);
VAR I:INTEGER;
BEGIN
  PENCOLOR(NONE);
  MOVETO(10*(LETTER MOD 16)+3,149-(LETTER DIV 16)*10);
  PENCOLOR(COLOR);
  TURNTO(0);
  FOR I:=1 TO 4 DO
  BEGIN
    MOVE(10); (* Quadrat zeichnen *)
    TURN(90);
  END;
  PENCOLOR(NONE);
END;

PROCEDURE CROSSHAIR(X,Y:INTEGER);
BEGIN
  PENCOLOR(NONE);
  MOVETO(201+X*10,83+Y*10);
  TURNTO(0);
  PENCOLOR(REVERSE);
  MOVE(4);
  PENCOLOR(NONE);
  MOVETO(203+X*10,81+Y*10);
  TURNTO(90);
  PENCOLOR(REVERSE);
  MOVE(4);
  PENCOLOR(NONE);
END;

PROCEDURE EXPAND(CH:CHARIMAGE);
VAR ROW,DOT:0..7;
BEGIN
  FOR ROW:= 0 TO 7 DO
   FOR DOT:= 0 TO 6 DO
    BEGIN
      MOVETO(200+DOT*10,80+ROW*10);
      IF DOT IN CH[ROW] THEN WCHAR(CHR(1))
       ELSE WCHAR(' ');
    END;
```

```
END;  (* EXPAND *)

PROCEDURE CLEARFIELD;
VAR ROW:0..7;
BEGIN
  FOR ROW:=0 TO 7 DO
   CURRENTCH[ROW]:=[];
  EXPAND(CURRENTCH);
END;

PROCEDURE SETUP;
VAR J:INTEGER;
BEGIN
    INITTURTLE;

    PRINTAT(0,180,'Pascal SYSTEM.CHARSET Editor version 2');
    PRINTAT(155,170,'von D. Rosenhain');
    PRINTAT(0,160,'Zeichensatz:'); (* KEIN ZEICHENSATZ *)

    DISPLAYSET;

    OLD:=0; I:=0;
    HILITE(I,WHITE);

    MOVETO(199,79); (* QUADRAT ZEICHNEN *)
    PENCOLOR(WHITE);
    MOVETO(268,79);
    MOVETO(268,159);
    MOVETO(199,159);
    MOVETO(199,79);
    PENCOLOR(NONE);

    FOR J:= 1 TO 7 DO (* GITTER ZEICHNEN *)
    BEGIN
        MOVETO(199,79+J*10);
        PENCOLOR(WHITE);
        MOVETO(268,79+J*10);
        PENCOLOR(NONE);
    END;

    FOR J:= 1 TO 6 DO
    BEGIN
        MOVETO(199+J*10,79);
        PENCOLOR(WHITE);
        MOVETO(199+J*10,159);
        PENCOLOR(NONE);
    END;
    CLEARFIELD;

    H:=0; V:=0;
    CROSSHAIR(H,V);
    OLDH:=H; OLDV:=V;
    ONE:=TRUE;

    CHARTYPE(5);
    PRINTAT(0,60,'Befehle :');
    CHARTYPE(10);
    PRINTAT(90,60,'weitere mit <Ret>...');
END;
```

```
PROCEDURE MENU1;
BEGIN
    VIEWPORT(0,279,0,59);
    FILLSCREEN(BLACK);

    CHARTYPE(5);
    PRINTAT(0,44,'Zeichenwahl:');

    CHARTYPE(10);
    PRINTAT(0,30,'W: hoch');
    PRINTAT(0,20,'Z: runter');
    PRINTAT(0,10,'A: links');
    PRINTAT(0, 0,'S: rechts');

    CHARTYPE(5);
    PRINTAT(100,44,'Editieren:');

    CHARTYPE(10);
    PRINTAT(100,30,'I: hoch');
    PRINTAT(100,20,'M: runter');
    PRINTAT(100,10,'J: links');
    PRINTAT(100, 0,'K: rechts');
    PRINTAT(175,40,'Y: setzen');
    PRINTAT(175,30,'N: loeschen');
    PRINTAT(175,20,'C: init.');
    PRINTAT(175,10,'G: holen');
    PRINTAT(175, 0,'P: schreiben');
    VIEWPORT(0,279,0,191);
END;

PROCEDURE MENU2;
BEGIN
    VIEWPORT(0,279,0,59);
    FILLSCREEN(BLACK);
    MOVETO(0,40);
    PRINTAT(0,40,'Q:ende  T:Satz schr. R:Satz les.');
    PRINTAT(10,24,'File laden  :....................');
    PRINTAT(10,9 ,'File speich.:....................');
    VIEWPORT(0,279,0,191);
END;


BEGIN  (* HAUPTPROGRAMM *)
    SETUP;
    MENU1;
    REPEAT
       READ(KEYBOARD,CHOICE);
       CASE CHOICE OF
       ' ':BEGIN
            IF ONE THEN MENU2
                   ELSE MENU1;
            ONE:= NOT ONE;
           END;

       'A':I:= (I+128-1) MOD 128;

       'S':I:= (I+1) MOD 128;

       'Z':I:= (I+16) MOD 128;
```

```
  'W':I:= (I+128-16) MOD 128;

  'G':BEGIN
        CURRENTCH:=CURRENTSET[I];
        EXPAND(CURRENTCH);
        CROSSHAIR(H,V);
      END;

  'C':BEGIN
        CLEARFIELD;
        CROSSHAIR(H,V);
      END;

  'P':BEGIN
        CURRENTSET[I]:=CURRENTCH;
        SHOWCHAR(I);
      END;

  'R':BEGIN
        IF ONE THEN MENU2;
        FILEIN;
        ONE:=FALSE;
      END;

  'T':BEGIN
        IF ONE THEN MENU2;
        FILEOUT;
        ONE:=FALSE;
      END;

  'Y':BEGIN
        CURRENTCH[V]:=CURRENTCH[V]+[H];
        MOVETO(200+H*10,80+V*10);
        WCHAR(CHR(1));
        CROSSHAIR(H,V);
      END;

  'N':BEGIN
        CURRENTCH[V]:=CURRENTCH[V]-[H];
        MOVETO(200+H*10,80+V*10);
        WCHAR(' ');
        CROSSHAIR(H,V);
      END;

  'I':V:=(V+1) MOD 8;

  'M':V:=(V+8-1) MOD 8;

  'J':H:=(H+7-1) MOD 7;

  'K':H:=(H+1) MOD 7;

END(* CASE *);

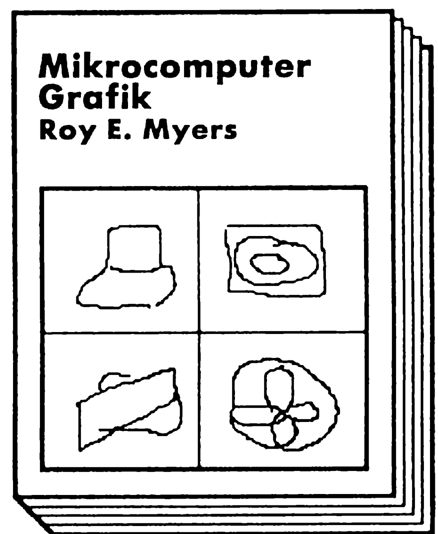
IF CHOICE IN ['I','J','K','M'] THEN
BEGIN
  CROSSHAIR(OLDH,OLDV);
  CROSSHAIR(H,V);
  OLDH:=H; OLDV:=V;
END;
```

```
      IF CHOICE IN ['W','A','S','Z'] THEN
      BEGIN
        HILITE(OLD,BLACK);
        HILITE(I,WHITE);
        OLD:=I;
      END;

    UNTIL CHOICE='Q';
END.
```

Weitere Bücher aus dem Pandabooks-Verlag:





Alles, was Sie bei APPLESOFT bisher vermißt haben:

- eine Input-Routine, mit der Sie auch Kommas und Doppelpunkte eingeben können
- ein PRINT USING Kommando für formatierte Ausgaben
- eine schnelle ''Garbage Collection''

und viele andere nützliche Utilities, dazu detaillierte Informationen über die Arbeitsweise des BASIC-Interpreters, Einsprungadressen, Systemvariablen.

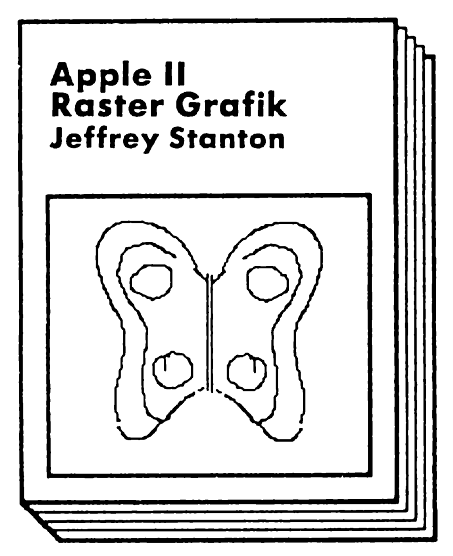
3-89058-033-5 ca. 250 S. DM 44,--
Diskette zum Buch DM 50,--

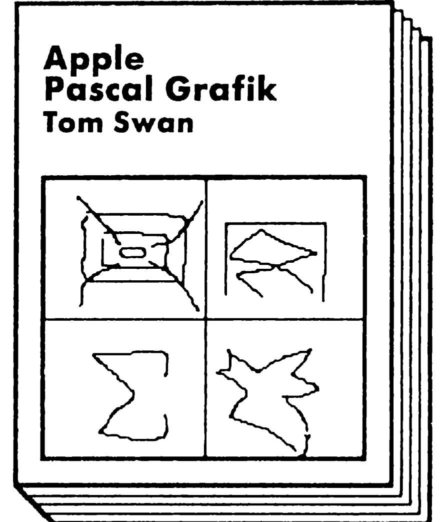
Endlich anspruchsvolle Apple-Grafik für BASIC-Programmierer.
Mikrocomputer Grafik

- enthält fast 80 lauffertige BASIC-Programme, die die beschriebenen Konzepte illustrieren.
- beschreibt "Hidden Line"- und "Hidden Surface"-Techniken, Skalierung, Rotation und Translation von Grafiken
- bietet eine Einführung in die Animationstechnik

3-89058-000-9 292 Seiten DM 49,--
Diskette zum Buch DM 50,--

Weitere Bücher aus dem Pandabooks-Verlag:





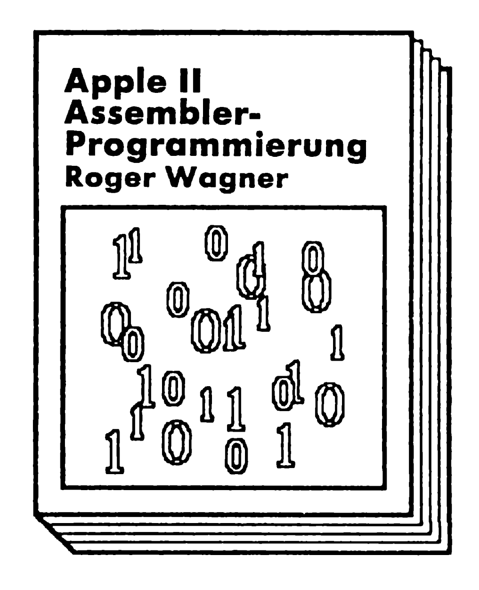
Die Qualität kommerzieller Arcadespiele läßt sich mit APPLESOFT BASIC alleine nicht erreichen. Jeffrey Stanton führt in die Eigenarten der hochauflösenden Apple-Grafik ein und präsentiert schließlich eine Reihe extrem schneller Assembler-Routinen. mit denen Sie viele Effekte bekannter Spiele selbst programmieren können. Gute BASIC-Kenntnisse werden vorausgesetzt, eine Einführung in Assembler-Programmierung wird gegeben.

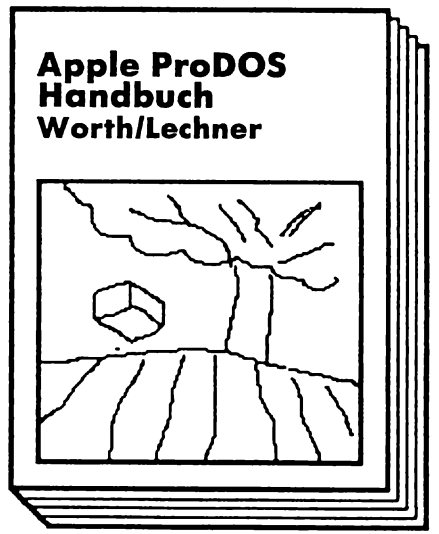
3-89058-006-8 299 Seiten DM 49,--
Diskette zum Buch DM 50,--

22 Pascal-Programme, mit denen Sie die Grafik-Möglichkeiten Ihres Apple voll ausschöpfen:
"Designer" ermöglicht es Ihnen, eigene Zeichensätze zu entwerfen.
"Gredit" unterstützt Sie beim Entwerfen komplexer Bildschirm-Grafiken
"Printfoto" bringt Ihre Entwürfe aufs Papier.
Darüberhinaus bietet das Buch eine Fülle fertiger Prozeduren, die Sie zeitsparend in Ihre eigenen Programme einbauen können

3-89058-009-2 280 Seiten DM 49,--
Diskette zum Buch DM 50,--

Weitere Bücher aus dem Pandabooks-Verlag:





Das Assembler-Lehrbuch für BASIC-Kenner.
Roger Wagner, der Autor vieler bekannter Software-Pakete, schrieb eine monatliche Kolumne über Assembler-Programmierung in der Apple-Zeitschrift SOFTALK.
Der vorliegende Band faßt diese Reihe, korrigiert und erweitert, zusammen: Eine stufenweise Einführung in die Befehle und Strukturen der 6502-Assemblersprache, mit vielen Beispielen von der einfachen Tongenerierung bis zum Diskettenzugriff.

3-89058-003-3 277 Seiten DM 48,--
Diskette zum Buch DM 50,--

Don Worth und Peter Lechner sind die Autoren von "Beneath Apple DOS", der Bibel aller DOS 3.3-Benutzer. In ihrem neuen Buch haben sie sich ProDOS vorgenommen und mit der ihnen eigenen Gründlichkeit zerlegt. Ausführliche Beschreibung der Arbeitsweise, der Dateitypen, des "MLI" und des "BASIC SYSTEM". Das Standardwerk für jeden ProDOS-Anwender.

3-89058-036-X 270 Seiten DM 46,--
Diskette zum Buch DM 50,--

Eine Sammlung von Utilities für den Pascal-Programmierer:
P-Code-Decodierer, Disk-Zapper, verbesserte I/O-Prozeduren, File-Konvertierer (Pascal nach DOS und zurück), Grafik-Routinen, Textformatierer, u.a.
Viele wichtige Informationen, Systemadressen, Tips und Tricks.

Pandabooks Berlin **ISBN 3-89058-030-0**